TUGAS AKHIR - KI141502

Rancang Bangun *Mini Synchronous Game* Food Fest dalam *Game* Sosial Food Merchant Saga pada Perangkat Android

Muhammad Riduwan
NRP 5110100144

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman H., S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT - KI141502

Design and implementation of Synchronous Mini Games Food Fest in Social Games Food Merchant Saga on Android Devices

Muhammad Riduwan
NRP 5110100144

Advisor
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman H., S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

LEMBAR PENGESAHAN

Rancang Bangun *Mini Synchronous Game* Food Fest dalam *Game* Sosial Food Merchant Saga pada Perangkat Android

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Interaksi, Grafik, dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Muhammad Riduwan
NRP : 5110 100 144

Disetujui oleh Dosen Pembimbing tugas akhir :

IMAM KUSWARDAYAN, S.Kom., M.T. .....
NIP: 19761215200312100 (pembimbing 1)

RIDHO RAHMAN H., S.Kom., M.Sc. .....
NIP: 198702132014041001 (pembimbing 2)

SURABAYA
JANUARI 2015

Rancang Bangun *Mini Synchronous Game* Food Fest dalam *Game* Sosial Food Merchant Saga pada Perangkat Android

Nama Mahasiswa	: Muhammad Riduwan
NRP	: 5110100144
Jurusan	: Teknik Informatika FTIf-ITS
Dosen Pembimbing 1	: Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing 2	: Ridho Rahman H., S.Kom., M.Sc.

ABSTRAK

Pada era digital saat ini teknologi game sudah mengalami perkembangan yang pesat. Game dapat dimainkan di berbagai perangkat elektronik yang dapat menunjang aktivitas manusia. Food merchant saga adalah game sosial yang bertemakan bisnis kuliner masakan Indonesia. Pemain memiliki sebuah pujasera dan mengelola beberapa kedai makanan.

Dalam game Food Merchant Saga terdapat mini game Food Fest. Mini game ini merupakan mode permainan online dengan pemain lain yang dapat berinteraksi secara synchronous. Proses sinkronisasi dikelola menggunakan web service, dimana pemain akan saling bertukar data. Setiap perubahan data antar pemain yang tergabung dalam satu festival akan langsung diketahui pemain lainnya. Mini game ini merupakan satu modul pendukung game utama Food Merchant Saga.

Pengujian dilakukan menggunakan metode blackbox dengan mengamati hasil dari fungsionalitas, integritas dan performa dari sistem. Sinkronisasi dapat berjalan dengan baik, namun banyaknya pemain akan mempengaruhi waktu respon dari sistem. Semua asset yang terhubung pada mini game akan diakumulasi dengan game utama. Sehingga dengan adanya mini game ini akan menjadi fitur lebih dalam game Food Merchant Saga.

Kata kunci: Game Sosial, Massively Multiplayer Online Game, MySQL, Sinkronisasi Data, Unity 2D, Web Service.

Design and implementation of Synchronous Mini Games Food Fest in Social Games Food Merchant Saga on Android Devices

Student Name	: Muhammad Riduwan
Student ID	: 5110100144
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Imam Kuswardayan, S.Kom., M.T.
Advisor 2	: Ridho Rahman H., S.Kom., M.Sc.

ABSTRACT

In today's digital era of gaming technology has been progressing rapidly. Games can be played in a variety of electronic devices that can support human activities. Food Merchants Saga is a social game themed Indonesian culinary business. The player has a food court and manages several food stalls.

In the Food Merchant Saga games are mini games Food Fest. This mini-game is a game mode online with other players that can interact synchronously. Synchronization process is managed using a web service, where players will exchange data. Any data changes between players who are members of the festival will be immediately known other players. This mini-game is a game support module main food merchants saga.

Tests carried out using blackbox method by observing the results of the functionality, integrity and performance of the system. Synchronization can be run well, but many players will affect the response time of the system. All assets are connected to the mini games are accumulated with the main game. So with the mini game will be more in-game features food merchants saga.

Keyword**: **Data Synchronous Process, Social Games, Massively Multiplayer Online Game, MySQL, Unity 2D, Web Service.

KATA PENGANTAR

Pertama-tama kami panjatkan puji syukur kehadirat Allah SWT atas berkat limpahan rahmatNya, kami dapat menyelesaikan tugas akhir yang berjudul :

"Rancang Bangun Mini Synchronous Game Food Fest dalam Game Sosial Food Merchant Saga pada Perangkat Android"

Harapan dari penulis semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi yang nyata.

Dalam usaha menyelesaikan tugas akhir ini penulis banyak mendapatkan bantuan dari berbagai pihak baik material maupun moril. Oleh karena itu penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT, karena tanpa ijin dari-Nya, penulis tidak akan mampu menyelesaikan tugas akhir ini.
2. Bapak Dariyo, Ibu Sumikah dan Fitria Anis Sushanti selaku keluarga tercinta penulis yang telah memberikan banyak doa dan dukungannya penulis berjanji akan selalu membahagiakan kalian.
3. Bapak Imam Kuswardayan dan Bapak Ridho Rahman selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan tugas akhir ini.
4. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
5. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
6. Teman-teman tim Food Merchant Saga, Aminudin Rahman, M Fadjar Rhomadoni, Muamar Agus Salim, Mochamad Yasin yang telah memberikan dukungan dan kerjasamanya.
7. Teman-teman eks pengawas Kopma dr.Angka ITS, Mas Fatkhur, Mas Nasrul, Iis, Amik dan Ema yang telah memberikan

banyak dukungan moril sehingga memberikan tambahan semangat untuk penulis menyelesaikan tugas akhir.

8. Teman-teman angkatan 2010 jurusan Teknik Informatika ITS yang telah menemani perjuangan selama 4 tahun ini atas saran, masukan, dan dukungan terhadap pengerjaan tugas akhir ini.
9. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Januari 2015

Muhammad Riduwan

DAFTAR ISI

LEMBAR PENGESAHAN v
ABSTRAK vii
ABSTRACT ix
KATA PENGANTAR xi
DAFTAR ISI xiii
DAFTAR GAMBAR xvii
DAFTAR TABEL xix
DAFTAR KODE SUMBER xxi
BAB I PENDAHULUAN 1
1.1. Latar Belakang 1
1.2. Tujuan 2
1.3. Rumusan Permasalahan 2
1.4. Batasan Permasalahan 3
1.5. Metodologi 3
1.6. Sistematika Penulisan 5
BAB II DASAR TEORI 7
2.1. Game Sosial 7
2.2. Sistem Operasi Android 8
2.3. Unity3D 9
2.4. Scene 10
2.5. Game Object 10
2.6. Mono Framework 11
2.7. MonoBehaviour 11
2.8. Proses Synchronous 12
2.9. Asynchronous dan Synchronous Game 12
2.10. Massively Multiplayer Online Game (MMOG) 13
2.11. Web Service 14
2.12. MySQL 14
BAB III ANALISIS DAN PERANCANGAN SISTEM 17
3.1. Analisis 17
3.1.1. Analisis Permasalahan 17
3.1.2. Deskripsi Umum Perangkat Lunak 18

3.1.3. Analisis Sistem 19
3.1.4. Spesifikasi Kebutuhan Perangkat Lunak 22
3.1.5. Skenario kasus penggunaan 24
3.2. Tahap Perancangan 35
3.2.1. Perancangan Antarmuka Pengguna 35
3.2.2. Perancangan Proses 39
3.2.3. Perancangan Permainan 43
BAB IV IMPLEMENTASI 47
4.1. Implementasi Antarmuka 47
4.1.1. Halaman Awal 48
4.1.2. Halaman Tampilan Daftar Festival 50
4.1.3. Halaman Membuat Festival Baru 53
4.1.4. Halaman Mengikuti Festival 55
4.1.5. Implementasi *Screen Gameplay* 58
4.2. Implementasi Proses 59
4.2.1. Implementasi Proses Menampilkan Daftar Festival 60
4.2.2. Implementasi Proses Membuat Festival Baru 61
4.2.3. Implementasi Proses Mengikuti Festival 62
4.3. Implementasi Gameplay 63
4.3.1. Data Pemain 63
4.3.2. Pertukaran Data pada Festival 65
BAB V PENGUJIAN DAN EVALUASI 69
5.1. Lingkungan Pengujian 69
5.2. Skenario Pengujian 69
5.2.1. Pengujian Fungsionalitas 70
5.2.2. Pengujian Integritas 83
5.2.3. Pengujian Performa 89
5.3. Evaluasi Pengujian 91
5.3.1. Evaluasi Pengujian Fungsionalitas 91
5.3.2. Evaluasi Pengujian Integritas 92
5.3.3. Evaluasi Pengujian Performa 92
BAB VI KESIMPULAN DAN SARAN 95
6.1. Kesimpulan 95
6.2. Saran 95

DAFTAR PUSTAKA 97
A. LAMPIRAN A Diagram Blok Food Merchant Saga 99
B. LAMPIRAN B Kode Sumber.....101
BIODATA PENULIS 137

DAFTAR TABEL

Tabel 3.1 Daftar Kebutuhan Fungsional Perangkat Lunak 23
Tabel 3.2 Keterangan Kasus Penggunaan pada Food Fest..... 25
Tabel 3.3 Spesifikasi Kasus Penggunaan Mengadakan Festival Makanan 26
Tabel 3.4 Spesifikasi Kasus Penggunaan Mengikuti Festival Makanan 28
Tabel 3.5 Spesifikasi Kasus Penggunaan Menampilkan Data Festival Aktif 30
Tabel 3.6 Spesifikasi Kasus Penggunaan Memperbarui Data ke *Server* pada Saat Festival Berlangsung 32
Tabel 3.7 Spesifikasi Kasus Penggunaan Mengolah Data dari *Server* pada Saat Festival Berlangsung 33
Tabel 5.1 Skenario Pengujian Meminta Data Festival yang Aktif 70
Tabel 5.2 Skenario Pengujian Membuat Festival Baru 73
Tabel 5.3 Skenario Pengujian Mengikuti Festival 75
Tabel 5.4 Skenario Pengujian Memperbarui Data ke *Server* pada Saat Festival Berlangsung 77
Tabel 5.5 Skenario Pengujian Mengolah Data dari *Server* pada Saat Festival Berlangsung 80
Tabel 5.6 Skenario Pengujian Integrasi dengan Modul AI 84
Tabel 5.7 Skenario Pengujian Integrasi dengan Asset Pemain pada Permainan Utama 85
Tabel 5.8 Spesifikasi Pengujian Performa 90
Tabel 5.9 Hasil Uji Coba Performa 90

DAFTAR GAMBAR

Gambar 2.1 Cafeland, *Game* Sosial Berbasis Facebook 7
Gambar 2.2 Berbagai Macam Perangkat Android 8
Gambar 2.3 Tampilan Halaman Unity3D 9
Gambar 2.4 *Game Object Main Camera* yang Ditampilkan pada Jendela Hierarchy 10
Gambar 2.5 Skema Sederhana dari Arsitektur Sistem Mono *Framework* 11
Gambar 3.1 Arsitektur Aplikasi 20
Gambar 3.2 Pola Hubungan Modul *Mini Game* dengan Modul Lain 21
Gambar 3.3 Diagram Kasus Penggunaan pada *Food Fest* 24
Gambar 3.4 Diagram Aktivitas Mengadakan Festival Makanan 27
Gambar 3.5 Diagram Aktivitas Mengikuti Festival Makanan 29
Gambar 3.6 Diagram Aktivitas Menampilkan Data Festival Aktif 31
Gambar 3.7 Diagram Aktivitas Mengupdate Data ke *Server* pada Saat Festival Berlangsung 33
Gambar 3.8 Diagram Aktivitas Mengolah Data dari *Server* pada Saat Festival Berlangsung 34
Gambar 3.9 Rancangan Tampilan Awal *Mini Game* 35
Gambar 3.10 Rancangan Tampilan Halaman Festival List 36
Gambar 3.11 Rancangan Tampilan Halaman Create New Festival 37
Gambar 3.12 Rancangan Tampilan Halaman Join Festival 38
Gambar 3.13 Rancangan Tampilan *Gameplay* 39
Gambar 3.14 Diagram Alir Menampilkan Daftar Festival 40
Gambar 3.15 Diagram Alir Membuat Festival Baru 41
Gambar 3.16 Diagram Alir Mengikuti Festival 42
Gambar 3.17 Diagram Alur Sinkronisasi 45
Gambar 4.1 Tampilan *Gameplay* pada Food Merchant Saga 47
Gambar 4.2 Tampilan Halaman Awal *Game* 48
Gambar 4.3 Tampilan Daftar Festival Aktif 51
Gambar 4.4 Tampilan Halaman Membuat Festival Baru 54
Gambar 4.5 Tampilan Halaman Mengikuti Festival. 55
Gambar 4.6 Tampilan Lokasi Festival 58
Gambar 4.7 Kedai Festival 67

Gambar 5.1 Isi Tabel Festival pada Basisdata *Server* 71
Gambar 5.2 Halaman Daftar Festival Tanpa Koneksi 72
Gambar 5.3 Halaman Daftar Festival dengan Koneksi 72
Gambar 5.4 Membuat Festival Baru 74
Gambar 5.5 Membuat Festival Baru 74
Gambar 5.6 Isi Tabel Festival yang Baru 75
Gambar 5.7 Pengisian Data Kedai Festival 76
Gambar 5.8 Pengisian Data Kedai Festival 76
Gambar 5.9 Isi Tabel Festival 78
Gambar 5.10 Uji Coba pada *Gameplay* Keadaan 1 78
Gambar 5.11 Isi kolom Status Pemain Keadaan 1 79
Gambar 5.12 Uji Coba *Gameplay* Keadaan 2 79
Gambar 5.13 Isi Kolom Status Pemain Keadaan 2 80
Gambar 5.14 Tampilan *Gameplay* pada Pemain 1 81
Gambar 5.15 Tampilan *gameplay* Pada Pemain 2 82
Gambar 5.16 Pemain 1 Melakukan Perubahan Data 82
Gambar 5.17 Pemain 2 Mengetahui Perubahan Data Pemain 1 83
Gambar 5.18 Perilaku NPC Saat Tidak ada Kedai Festival 84
Gambar 5.19 Perilaku NPC pada Saat Ada Kedai Festival 85
Gambar 5.20 *Gameplay* Permainan Utama dengan 1 Kedai 87
Gambar 5.21 Mengikuti Festival dengan Kedai 1 87
Gambar 5.22 *Gameplay* Permainan Utama dengan 2 Kedai 88
Gambar 5.23 Mengikuti Festival dengan Kedai 2 88
Gambar 5.24 Halaman Depan Loadfocus 89
Gambar 5.25 Diagram Waktu Respon Sinkronisasi 93
Gambar A.1 Diagram Blok Bagian Pertama Aplikasi Permainan Food Merchant Saga Bagian Pertama 99
Gambar A.2 Diagram Blok Bagian Pertama Aplikasi Permainan Food Merchant Saga Bagian Kedua 100

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Pada era digital saat ini teknologi *game* sudah mengalami perkembangan yang pesat. *Game* atau permainan video yang pada awal penemuannya hanya dapat dimainkan pada sebuah komputer dengan teknologi grafis yang sederhana, kini telah mengalami banyak kemajuan seiring dengan berkembangnya teknologi baru.

Pada tahun 1977 Atari mengeluarkan *Video Computer system* yang lebih dikenal dengan nama Atari 2600 [1]. Produk ini mengawali era permainan video dengan sebuah perangkat *console game* yang dapat dimainkan di dalam rumah. Kemudian setelah jaringan internet secara umum dipakai oleh masyarakat, muncullah berbagai jenis *game* yang ada di internet atau lebih dikenal dengan sebutan *online game*. Lalu pada saat ini berkembanglah teknologi berbasis *mobile*. Perkembangan teknologi *mobile* mencakup berbagai aspek baik dari segi perangkat keras maupun perangkat lunak. Perangkat *mobile* dalam bentuk *smartphone* atau *tablet computer* sudah menjadi suatu kebutuhan bagi masyarakat. Oleh karena itu, berkembang pula *game* berbasis *mobile* dengan berbagai macam *genre* dan teknologi pendukungnya.

Pada saat ini, *game* tidak hanya sebatas permainan di suatu perangkat yang dimainkan satu atau dua orang pada satu waktu, namun *game* bisa dimainkan oleh banyak orang dalam waktu yang tak terbatas. *Game* yang mendukung interaksi antar banyak pemain disebut *game* sosial. Banyak unsur sosial di masyarakat yang dapat dijadikan sebagai *game*, seperti aktivitas jaman kerajaan, aktivitas jual beli, aktivitas bercocok tanam.

Android merupakan sebuah sistem operasi perangkat *mobile* untuk telepon seluler seperti telepon pintar dan komputer tablet. Pada toko aplikasi Android terdapat berbagai macam *game*. Namun disana belum ada *game* sosial buatan Indonesia yang secara khusus bertemakan bisnis kuliner. Oleh karena itu, dalam tugas akhir ini penulis akan mengembangkan sebuah *game* sosial bertemakan bisnis kuliner khas Indonesia yang dinamakan Food Merchant Saga. Karena disamping memberikan sarana hiburan yang positif, diharapkan *game* ini juga dapat menjadi sarana pengenalan masakan Indonesia.

Penulis secara khusus mengembangkan *game* ini pada modul *mini synchronous game* Food Fest yang merupakan mode permainan *online* pada *game* Food Merchant Saga. Pemain dengan pemain lain dapat membuka kedai untuk berdagang dan saling bersaing untuk mendapatkan keuntungan yang sebesar-besarnya.

1.2. Tujuan

Tugas Akhir ini memiliki tujuan untuk menghasilkan *mini synchronous game* Food Fest dalam *game* sosial Food Merchant Saga pada perangkat Android.

1.3. Rumusan Permasalahan

Beberapa permasalahan yang digunakan sebagai rumusan masalah adalah sebagai berikut.

1. Bagaimana mengatur sinkronisasi setiap alur permainan untuk ditampilkan secara *realtime*.
2. Bagaimana merancang aturan main yang disesuaikan dengan keadaan sinkronisasi dan batasan maksimum jumlah pemain.
3. Bagaimana mengintegrasikan modul *mini synchronous game* dengan modul lain dalam sistem *game Food Merchant Saga*.
4. Bagaimana implementasi *mini synchronous game Food Fest* dalam *game Food Merchant Saga* menggunakan C# pada lingkungan pengembangan Unity untuk perangkat *mobile*.

1.4. Batasan Permasalahan

Batasan lingkup masalah yang dibahas pada Tugas Akhir ini adalah sebagai berikut.

1. Aplikasi permainan ini dibangun menggunakan *tools* Unity dengan bahasa pemrograman C#.
2. Jumlah maksimal pemain yang dapat bergabung dalam satu *room Food Fest* adalah 10 pemain.

1.5. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir ini yaitu.

1. **Studi literatur**

   Pada tahap ini dilakukan pengumpulan informasi yang diperlukan untuk perancangan sistem yang meliputi informasi-informasi sebagai berikut.

   1. Perencanaan dan pembangunan perangkat lunak
   2. Penggunakan *middleware* Unity3D dengan tampilan 2 dimensi
   3. Penerapan konsep MMOG (*Massively Multiplayer Online Game)* dalam menangani sinkronisasi aktivitas pemain
   4. Penerapan konsep basis data dan *web service* dalam menangani alur data

2. **Analisis dan Perancangan Sistem**

   Pada tahap ini dilakukan analisa awal dan mendefinisikan kebutuhan sistem untuk mengetahui masalah yang sedang dihadapi. Dari proses tersebut selanjutnya dirumuskan rancangan sistem yang dapat memberi permecahan masalah tersebut. Langkah-langkah pada tahap ini adalah sebagai berikut.

   1. Analisis sistem dan aktor yang terlibat.
   2. Perancangan *use case diagram,* yang merupakan analisis kebutuhan pada aplikasi yang akan dibangun.
   3. Perancangan aplikasi dan *gameplay.*

3. Implementasi

Pada tahap ini dilakukan pembuatan perangkat lunak yang merupakan implementasi dari rancangan yang telah dibuat sebelumnya.

Perincian tahap ini adalah sebagai berikut.

1. Implementasi pembuatan antar muka
2. Implementasi proses
3. Implementasi *gameplay*

4. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan skenario yang telah disiapkan sebelumnya. Uji coba dan evaluasi dilakukan untuk mencari masalah yang mungkin timbul, mengevaluasi jalannya program, dan mengadakan perbaikan jika ada kekurangan. Pengujian dilakukan menggunakan pengujian kotak hitam (*blackbox*). Pengujian kotak hitam adalah pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, penguji dapat mendefinisikan kumpulan kondisi masukan dan melakukan pengujian pada spesifikasi fungsional program. Tahapan-tahapan dari uji coba yang akan dilakukan adalah sebagai berikut.

1. pengujian kebenaran (*correctness testing*) pada fungsi-fungsi yang terdapat di dalam sistem.
2. pengujian modularitas dan integrasi (*modularity & integrity testing*). Pengujian ini dilakukan untuk mengetahui apakah modul dapat diuji dan dijalankan secara terpisah tanpa terintegrasikan dengan modul yang lainnya maupun ketika diintegrasikan dengan modul lainnya untuk membentuk kesatuan *game* sosial Food Merchant Saga. Hal ini diperlukan untuk kemungkinan jika modul-modul lain belum selesai dikembangkan.

5. **Penyusunan buku tugas akhir**
   Pada tahap ini melakukan pendokumentasian dan laporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan tugas akhir. Buku tugas akhir ini bertujuan untuk memberikan gambaran dari pengerjaan tugas akhir ini dan diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut

1.6. Sistematika Penulisan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan
Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan tugas akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

Bab II Dasar Teori
Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan tugas akhir ini.

Bab III Analisis dan Perancangan Sistem
Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan data, arsitektur, proses dan perancangan antarmuka pada kakas.

Bab IV Implementasi
Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi
Bab ini membahas pengujian dengan metode pengujian

subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian hasil analisis perangkat lunak.

Bab VI Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan tugas akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir. Teori-teori tersebut meliputi *game* sosial, sistem operasi Android, Unity3D, *Scene*, *Game Object*, Mono *Framework*, MonoBehaviour, proses *synchronous*, *asynchronous* dan *synchronous game*, *Massively Multiplayer Online Game* (MMOG), *Web Service*, MySQL.

2.1. *Game* Sosial

Game sosial adalah permainan yang dimainkan dengan teman-teman yang berada pada suatu jaringan sosial sehingga pemain bisa mengundang atau mengajak teman untuk melakukan aktivitas tertentu dalam *game* [2]. *Game* sosial berkembang pesat seiring dengan berkembangnya media sosial.

Gambar 2.1 Cafeland, *Game* Sosial Berbasis Facebook

Gambar diatas adalah *game* Cafeland, merupakan salah satu contoh *game* sosial yang terkenal banyak dimankan oleh pengguna Facebook. Dengan adanya *game* sosial membuat aktivatas permainan dalam suatu *game* menjadi tak terbatas oleh tempat dan waktu, karena di mana pun dan kapan pun orang bisa bermain dan berinteraksi dengan menggunakan media *game*.

2.2. Sistem Operasi Android

Android merupakan sebuah sistem operasi yang berbasis Linux untuk telepon seluler seperti telepon pintar dan komputer tablet. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam piranti bergerak.

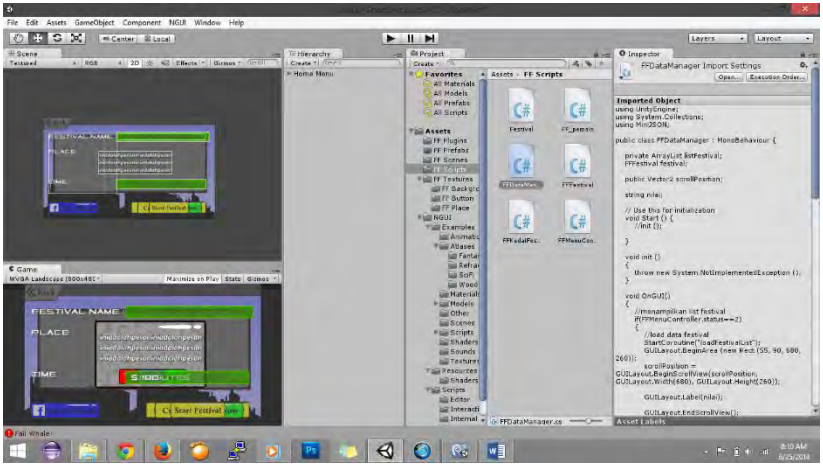
Awalnya, Google Inc. membeli Android Inc., pendatang baru yang membuat piranti lunak untuk ponsel. Kemudian untuk mengembangkan Android dibentuklah Open Handset Alliance, konsorsium dari 34 perusahaan peranti keras, piranti lunak, dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan NVidia. Pada saat perilisan perdana Android, 5 November 2007, Android bersama Open Handset Alliance menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak, Google merilis kode–kode Android di bawah lisensi Apache, sebuah lisensi perangkat lunak dan standar terbuka perangkat seluler [3].

Gambar 2.2 Berbagai Macam Perangkat Android

Gambar diatas menunjukan berbagai macam perangkat berbasis Android. Android selalu mengalami perkembangan dan pembaruan system setiap tahunnya. Saat ini sudah dikembangkan versi terbaru dari sistem operasi ini yaitu Android 5 dengan kode "Lollipop".

2.3. Unity3D

Unity3D merupakan sebuah ekosistem pengembangan permainan yang terintegrasi dan kaya akan alat atau perlengkapan yang sangat berguna untuk membangun permainan interaktif seperti pencahayaan, efek khusus, animasi, dan mesin fisika. Unity juga dapat digunakan untuk melakukan perubahan maupun menguji secara bersamaan pada permainan yang dibuat. Ketika sudah siap permainan dapat dipublikasikan ke berbagai macam perangkat yang mendukung seperti Mac, PC, Linux, Windows Store, Windows Phone 8, Android, Blackberry 10, Wii U, PS3, dan Xbox 360 [4].



Gambar 2.3 Tampilan Halaman Unity3D

Gambar diatas menunjukkan tampilan halaman Unity3D, dimana terdapat banyak panel yang membantu dalam proses

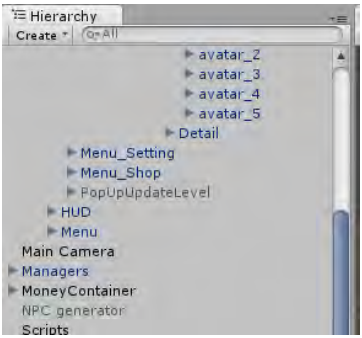
pembuatan *game*. Unity3D juga dapat digunakan untuk membuat *game* dengan tampilan secara 2 dimensi. Hal itu dapat dilakukan dengan mengubah tampilan scene secara 2D dan proyeksi kamera dari *perspective* menjadi *orthographic*.

2.4. *Scene*

Setiap *object* di dalam permainan yang dibuat oleh Unity diletakkan dalam sebuah *scene. Scene* dapat digunakan untuk membuat menu utama, level, dan lain sebagainya. Dalam mendesain permainan dengan Unity, dapat dianalogikan jika sebuah *scene* sama artinya dengan sebuah level. Di dalam setiap *scene,* penulis meletakkan entitas, halangan serta dekorasi permainan untuk dibangun menjadi sebuah permainan video.

2.5. *Game Object*

Game Object merupakan bagian terkecil yang ada pada sebuah permainan yang dibagun menggunakan Unity. Programer diperbolehkan menambahkan atribut, serta *class program* ke dalam sebuah *game object*. Setiap *game object* pada unity memiliki fungsi yang berbeda, misalnya pada *game object main camera* seperti yang ditunjukkan pada Gambar 2.4. Main camera merupakan objek yang mengatur posisi dan pengambilan gambar oleh kamera.

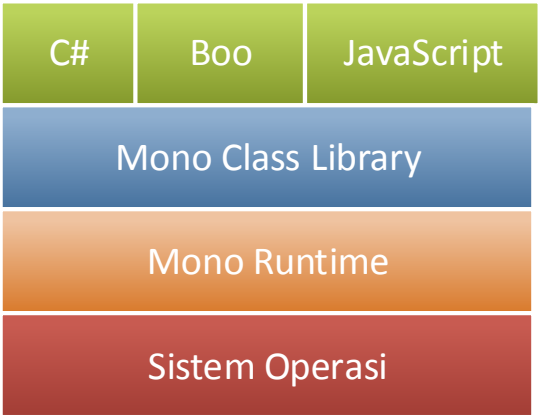


Gambar 2.4 *Game Object Main Camera* yang Ditampilkan pada Jendela Hierarchy

2.6. Mono *Framework*

Mono merupakan *development framework* yang setara dengan framework .NET yang juga merupakan *development framework* yang dibuat oleh Microsoft. Atau juga setara dengan framework java yang disebut dengan Java Platform.

Development framework biasanya menyediakan suatu SDK (*Software-development kits*) tertentu untuk membantu dalam pemrograman. Sedangkan setiap SDK juga menyediakan sebuah *compiler* tersendiri untuk mengeksekusi program yang dibuat. Pada Mono,Java, dan .NET juga menyediakan suatu *runtime* khusus, pada Mono *Framework* disebut Mono *Runtime*, jadi pada dasarnya program tidak dijalankan langsung oleh prosesor namun program dijalankan oleh sebuah *runtime* tersendiri yang sesuai dengan arsitektur dari prosesor. Inilah yang membuat Unity dapat mendukung berbagai macam *platform* yang berbeda.



Gambar 2.5 Skema Sederhana dari Arsitektur Sistem Mono *Framework*

2.7. MonoBehaviour

MonoBehaviour merupakan kelas induk dari sebuah *game object* yang ada pada Unity. Sehingga setiap kelas program yang

akan di implementasikan ke sebuah *game object* harus merupakan turunan dari MonoBehaviour. Jika program menggunakan javascript maka setiap kelas yang dibuat secara otomatis merukanan turunan dari MonoBehaviour, ini berbeda jika kelas yang dibuat menggunakan bahasa C# atau Boo dimana program harus secara eksplisit diturunkan dari MonoBehaviour.

Pada checkbox yang ada pada editor akan mengatur berjalannya fungsi Start(), Awake(), Update(), FixedUpdate(), serta OnGUI(), dan tidak akan mempengaruhi berjalannya fungsi dasar lain yang ada pada MonoBehaviour.

2.8. Proses *Synchronous*

Sebuah proses *synchronous* dipanggil oleh operasi *request*/*response*, lalu hasil dari proses akan dikembalikan ke pemanggil sesegera mungkin melalui operasi tersebut [5]. Proses pengiriman dan penerimaan diatur sedemikian rupa sehingga memiliki pengaturan yang sama. Umumnya, pengaturan ini didasarkan pada waktu dalam mengirimkan sinyal. Waktu ini diatur oleh denyut listrik secara periodik yang disebut *clock*. Dengan kata lain *synchronous* adalah sistem operasi untuk kejadian yang terjadi pada waktu bersamaan, berkelanjutan, dan dapat diprediksi, contohnya adalah chatting

2.9. *Asynchronous* dan *Synchronous Game*

Proses *asynchronous* merupakan kebalikan dari proses *synchronous* yang berarti bahwa sebuah proses komunikasi data yang tidak bergantung pada waktu yang tetap. Proses pertukaran data secara *asynchronous* dan *synchronous* banyak diterapkan dalam dunia *game* khususnya *game* online. Dalam *game asynchronous*, pemain berinteraksi dengan AI (*Artificial Intelligence*). Sedangkan pada *game synchronous* peran AI lebih tergantikan dengan pemain lain [6]. Contoh dari *game asynchronous* adalah Candy Crush Saga, Happy Mall Story. Contoh dari *game synchronous* adalah Dota (Defense of The Ancients).

2.10. *Massively Multiplayer Online Game (MMOG)*

Massively Multiplayer Online Game (MMOG) mengacu pada video *game* yang memungkinkan banyak pemain untuk berpartisipasi secara simultan melalui koneksi internet [7]. Hal ini berarti pemain satu dengan pemain lain dapat saling berinteraksi, baik secara langsung maupun tidak langsung. Dalam sebuah *game* MMOG, pemain yang berinteraksi dapat mencapai ratusan maupun ribuan. Oleh karena itu, kemampuan *server* dalam melayani *client* sangat penting dan menentukan kualitas suatu *game*.

Terdapat banyak jenis dari *Massively Multiplayer Online Game* (MMOG), beberapa diantaranya adalah sebagai berikut.

1. MMORPG
   MMORPG merupakan singkatan dari *massively multiplayer online role playing game*. Jenis *game* ini mempunyai kemampuan untuk mengelola ribuan orang untuk bermain bersama pada sebuah *server* tunggal [8]. Pada umumnya jenis permainan yang dijalankan seputar kehidupan tentang ekonomi, pekerjaan, tantangan, perang dan sebagainya. Biasanya tidak terdapat akhir dari permainan, sehingga membuat pemain senantiasa bermain sangat lama bahkan mereka dapat saling bersahabat dan membentuk suatu interaksi sosial melalui kehidupan online dalam *game*. Contoh dari MMORPG adalah World of Warcraft.
2. MMO First-Person Shooters
   MMOFPS adalah permainan *multiplayer* dengan area tempur yang besar, dimana pemain dapat bermain secara tunggal atau bergabung dengan pemain lainnya untuk menjadi suatu tim. *Game* jenis ini menggunakan sudut pandang orang pertama. Contoh dari *game* ini adalah World War II Online.
3. MMO Real-Time Strategy *Game*s
   MMORTS menggabungkan konsep *real-time strategy* (RTS) dengan permainan yang terus berjalan pada suatu permainan online. Pemain harus bisa merebut, mempertahankan atau melakukan apapun sesuai misi yang dijalankan. Contoh dari *game* ini adalah Heros of Newerth, Dota, Avalon.

2.11. Web Service

Web Service adalah layanan yang dibuat dan disediakan dari *web server* bisnis untuk pengguna web atau program yang terhubung [9]. Secara umum, web service dapat diidentifikasikan dengan menggunakan URL seperti hanya web pada umumnya. Namun yang membedakan *web service* dengan web pada umumnya adalah interaksi yang diberikan oleh *web service*. Berbeda dengan URL web pada umumnya, URL *web service* hanya menggandung kumpulan informasi, perintah, konfigurasi atau sintaks yang berguna membangun sebuah fungsi-fungsi tertentu dari aplikasi.

Web service dapat diartikan juga sebuah metode pertukaran data, tanpa memperhatikan dimana sebuah basisdata ditanamkan, dibuat dalam bahasa apa sebuah aplikasi yang mengkonsumsi data, dan di *platform* apa sebuah data itu dikonsumsi. *Web service* mampu menunjang interoperabilitas. Sehingga *web service* mampu menjadi sebuah jembatan penghubung antara berbagai sistem yang ada.

2.12. MySQL

MySQL adalah sistem manajemen basisdata SQL yang bersifat terbuka. Sistem basisdata MySQL mendukung beberapa fitur seperti *multithreaded, multi-user*, dan SQL *database management system* (DBMS). Basisdata ini dibuat untuk keperluan sistem basisdata yang cepat, handal dan mudah digunakan.

Ulf Micheal Widenius adalah penemu awal versi pertama MySQL yang kemudian pengembangan selanjutnya dilakukan oleh perusahaan MySQL AB. MySQL AB yang merupakan sebuah perusahaan komersial yang didirikan oleh para pengembang MySQL. MySQL sudah digunakan lebih dari 11 millar instalasi saat ini. Berikut ini beberapa kelebihan MySQL sebagai basisdata *server* antara lain :

- *Source* MySQL dapat diperoleh dengan mudah dan gratis.
- Sintaksnya lebih mudah dipahami dan tidak rumit.
- Pengaksesan database dapat dilakukan dengan mudah.
- MySQL merupakan program yang *multithreaded*, sehingga dapat dipasang pada *server* yang memiliki multiCPU.

- Didukung program-program umum seperti C, C++, Java, Perl, PHP, Python, dsb.
- Bekerja pada berbagai *platform*. (tersedia berbagai versi untuk berbagai sistem operasi).
- Memiliki jenis kolom yang cukup banyak sehingga memudahkan konfigurasi sistem database.
- Memiliki sistem sekuriti yang cukup baik dengan verifikasi *host*.
- Mendukung ODBC untuk sistem operasi Windows.
- Mendukung record yang memiliki kolom dengan panjang tetap atau panjang bervariasi.

MySQL dan PHP merupakan sistem yang saling terintegrasi. Maksudnya adalah pembuatan basisdata dengan menggunakan sintak PHP dapat dibuat. Sedangkan input yang di masukkan melalui aplikasi web yang menggunakan *script serverside* seperti PHP dapat langsung dimasukkan ke basisdata MySQL yang ada di *server* dan tentunya web tersebut berada di sebuah *web server*.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tahap analisis permasalahan dan perancangan dari sistem yang akan dibangun. Analisis permasalahan membahas permasalahan yang yang diangkat dalam pengerjaan tugas akhir. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat. Pendekatan yang dibuat dalam perancangan ini adalah pendekatan berorientasi objek. Perancangan direpresentasikan dengan diagram UML (*Unified Modelling Language*).

3.1. Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain analisis permasalahan, deskripsi umum sistem, kasus penggunaan sistem, dan kebutuhan perangkat lunak.

3.1.1. Analisis Permasalahan

Permasalahan utama yang diangkat dalam pembuatan tugas akhir ini adalah bagaimana cara merancang dan membangun *mini game* yang merupakan salah satu fitur dari *game* utama Food Merchant Saga. *Mini game* tersebut dapat dimainkan dengan banyak pemain yang terintegrasi secara online dan menggunakan alur sinkronus.

Setiap perubahan yang terjadi pada pemain akan diperbarui secara berkala dan akan dikembalikan pada pemain lainnya sebagai suatu informasi. Basisdata akan diletakkan pada aplikasi *server*. Komunikasi antara *server* dan *client* akan menggunakan suatu *web service*.

Mini game tersebut harus bisa mengakses data pada *game* utama Food Merchant Saga dan data beberapa pemain yang dibutuhkan di basisdata *server*. Selain itu *mini game* tersebut harus bisa memberikan unsur *fun* pada pemain karena hal

tersebut merupkan suatu keharusan untuk jenis aplikasi permainan.

3.1.2. Deskripsi Umum Perangkat Lunak

Pada tugas akhir ini, akan dibuat suatu *game* sosial tentang bisnis kuliner Indonesia yang berjudul Food Merchant Saga. Namun penulis hanya berfokus pada sisi *mini game* dari Food Merchant Saga. *Mini game* ini yang selanjutnya akan disebut dengan Food Fest, merupakan suatu fitur permainan online dalam *game* Food Merchant Saga. Pemain dapat berjualan produk kuliner mereka sendiri secara bersama-sama yang tergabung dalam suatu festival kuliner.

Berikut akan dijelaskan deskripsi umum dari permainan utama pada Food Merchant Saga dan juga mode permainan online pada Food Fest.

3.1.2.1. Food Merchant Saga

Game Food Merchant Saga adalah *social mobile game* yang bergenre simulasi berbisnis kuliner indonesia dalam sebuah Pujasera. Pemain akan diberi modal berupa sebuah Pujasera kecil yang hanya berisi kedai bakso dan kedai nasi pecel. Kemudian seiring dengan meningkatnya level pujasera, maka luas bangunan Pujasera juga akan bertambah. Pemain bisa membeli kedai baru dan bisa memproduksi menu makanan baru.

Pemain bisa meningkatkan kapasitas produksi makanan dan menambah jumlah jenis dan variasi makanan di setiap kedai. Syarat untuk mendapatkan poin yang berguna untuk naik level adalah menyelesaikan daftar tugas yang tersedia. Pemain akan mendapatkan uang dari setiap penjualan makanan dan penyelesaian misi. *Game* Food Merchant Saga memiliki fitur sosial yaitu mengunjungi Pujasera teman, memberikan hadiah kepada teman, dan mengundang teman untuk berkunjung ke Pujasera kita. *Game* ini juga dilengkapi *mini synchronous game* tentang festival makanan yang disebut Food Fest. Gambar detail

mengenai fitur dan fungsi yang terdapat pada Food Merchant Saga dapat dilihat pada diagram blok yang ada di Lampiran 1.

3.1.2.2. Food Fest

Food Fest adalah sebuah *mini game* dalam *game* utama Food merchant Saga. Dalam Food Fest, pemain akan berjualan produk kuliner yang dimilikinya secara bersama-sama dengan pemain lain. Pemain menjajakan makanan yang tersedia dalam kedainya.

Ketika mengakses Food Fest, pemain akan ditampilkan daftar festival yang tersedia. Pemain dapat memilih bergabung pada salah satu festival, atau membuat festival sendiri. Ketika pemain bergabung pada salah satu festival, maka dia harus melengkapi data-data yang diperlukan, seperti nama kedai, jenis kedai. Setelah pemain terdaftar pada suatu festival, maka pemain harus menunggu sampai jadwal festival tersebut dimulai.

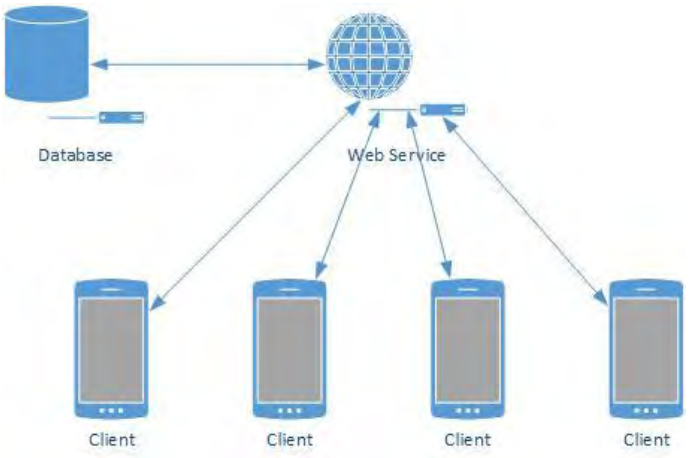
Jika pemain memilih untuk membuat festival sendiri, maka pemain juga harus memasukkan data festival yang diperlukan. Setelah itu, pemain otomatis terdaftar pada festival yang dibuat. Pemain hanya dapat mengikuti satu festival yang aktif. Pemain dapat mengikuti festival yang lain ketika festival yang diikuti sudah berakhir.

Pada saat festival dimulai, pemain harus menjaga stand yang dimilikinya. Pembeli akan berkeliling tempat festival. Jika salah satu pembeli membeli produk makanan pemain, maka pemain harus menekan uang yang muncul dan pemain tersebut memperoleh penghasilan. Total penghasilan yang didapat pada festival akan diakumulasi dengan total uang yang didapat pemain.

3.1.3. Analisis Sistem

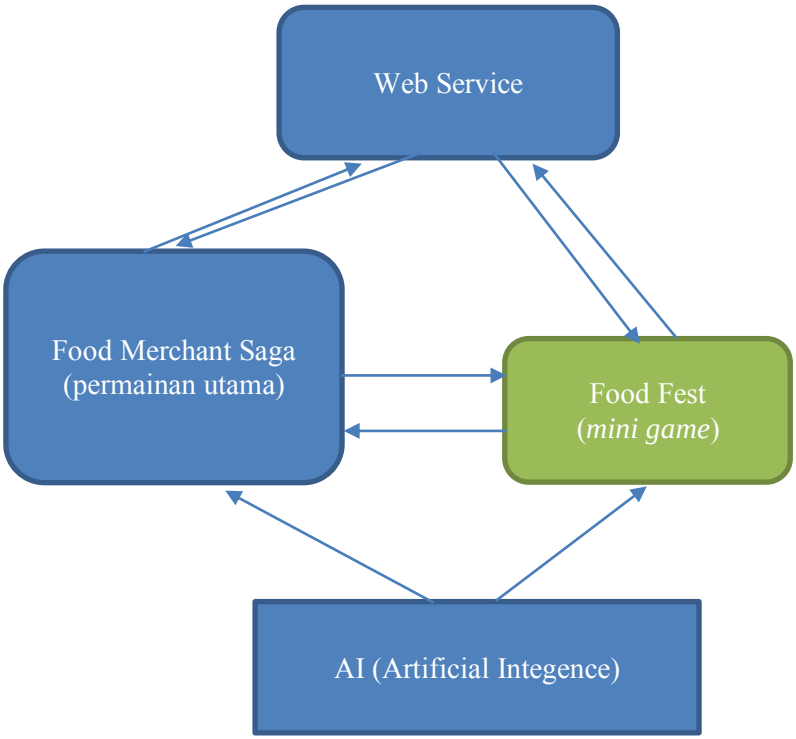
Pada mode permainan online Food Fest, terdapat sebuah sistem yang memiliki tiga lapisan utama. Tiga lapisan tersebut adalah lapisan aplikasi, lapisan web service, dan lapisan

basisdata. Lapisan aplikasi adalah sebuah lapisan dimana terdapat proses logika serta aset aplikasi yang dibutuhkan dalam *gameplay*. Lapisan ini akan meminta data dari basis data *server* melalui mekanisme web service. Selain itu setiap perubahan yang terjadi pada obyek-obyek dalam *game* akan diupdate ke basisdata secara berkala. Pada lapisan web service dan basisdata menangani proses pada *game* secara keseluruhan. Gambar 3.1 menunjukkan arsitektur aplikasi pada Food Fest.



Gambar 3.1 Arsitektur Aplikasi

Food Fest yang merupakan salah satu fitur dari *game* utama Food Merchant Saga, berhubungan dengan beberapa modul lain. Modul tersebut adalah modul *web service* dan modul AI (*Artificial Intelegence*), serta berhubungan langsung dalam pengaksesan data pada *game* utama Food Merchant Saga. Hubungan tersebut dapat dilihat pada Gambar 3.2.



Gambar 3.2 Pola Hubungan Modul *Mini Game* dengan Modul Lain

Pada Gambar 3.2, terdapat empat modul yang merupakan modul penyusun *game* Food Merchant Saga. Yang pertama adalah modul permainan utama Food Merchant Saga. Modul ini menangani seluruh aktivitas permainan utama yang meliputi elemen permainan seperti pemain, kedai, transaksi, inventori, dan menu sosial. Aktivitas pada modul permainan utama didukung oleh tiga modul lainnya, yaitu web service, AI (Artificial Integence), dan *mini game*.

Modul web service menangani penyimpanan data yang terintegrasi secara *online.* Seluruh data pada elemen permainan akan

disimpan pada basisdata *server*. Kemudian modul AI digunakan untuk membangkitkan karakteristik NPC (*Non-Playable Character*). NPC ini berperan sebagai pembeli pada *gameplay* utama maupun *mini game*.

Dan terakhir adalah modul *mini game*. Modul ini merupakan pendukung modul utama Food Merchant Saga. Tujuan adanya modul ini adalah untuk menambah daya tarik terhadap game. Hal ini dikarenakan pemain dapat bermain dengan pemain lainnya secara bersamaan serta pemain dapat menambah jumlah pemasukkan dari festival tersebut. Modul ini mengakses data pemain, kedai, dan transaksi kedai dari modul utama. Kemudian untuk menjalankan proses sinkronisasi, modul ini membutuhkan *web service* untuk tempat penampungan data. Pada saat festival berlangsung, data tersebut akan diolah dan dibagikan pada pemain lain. Sehingga akan ada data dan informasi yang sama antar pemain. Kemudian seperti pada modul permainan utama, modul AI juga berperan untuk membangkitkan karakteristik NPC pada *gameplay mini game* Food Fest.

3.1.4. Spesifikasi Kebutuhan Perangkat Lunak

Bagian ini berisi semua kebutuhan perangkat lunak yang diuraikan secara rinci seperti kebutuhan perangkat lunak dalam sistem yang mencakup kebutuhan fungsional dan kebutuhan non fungsional.

3.1.4.1. Kebutuhan Fungsional

Kebutuhan fungsional berisi proses-proses yang harus dimiliki sistem. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan dan reaksi sistem terhadap masukan atau pada situasi tertentu. Daftar kebutuhan fungsional dapat dilihat pada Tabel 3.1.

Tabel 3.1 Daftar Kebutuhan Fungsional Perangkat Lunak

Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
F-0001	Mengirimkan data ke *Server*	Segala aktivitas pemain baik membuat data baru atau mengupdate data yang telah ada akan dikirim ke *server* untuk disimpan dalam system basis data
F-0002	Menerima serta mengolah data dari *server*	Data dari *server*, baik itu di minta atau tidak oleh *client* akan diolah sesuai dengan kebutuhan

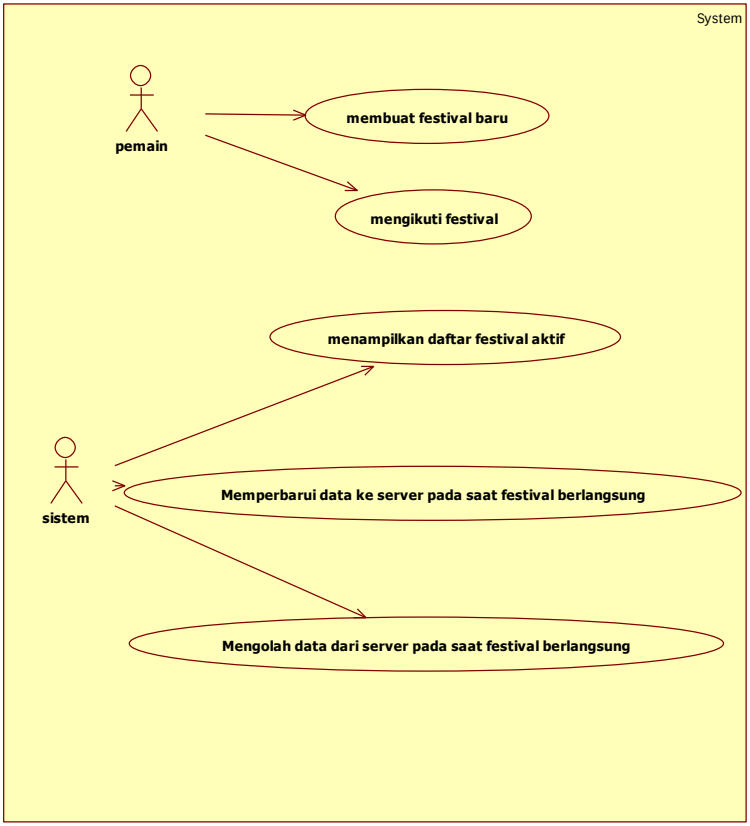
3.1.4.2. Kebutuhan Non Fungsional

Terdapat beberapa kebutuhan non-fungsional yang apabila dipenuhi, dapat meningkatkan kualitas dari permainan ini. Berikut daftar kebutuhan non-fungsional.

1. *Frame Rate*
   Permainan ini harus mampu dimainkan secara lancar, tidak ada lag dan nyaman di mata. Sebagian permainan 2D biasanya optimal pada Frame Rate 24-30 fps (*Frame per Second*) namun untuk animasi dan permainan 3D akan sangat baik jika *Frame Rate* bisa bertahan di sekitar 60 fps.
2. Kebutuhan Grafis
   Kenyamanan bermain berbanding lurus dengan kualitas grafis yang disajikan dalam permainan. Efek seperti shadow, shading dan texture merupakan salah satu daya tarik dalam permainan 3D. Efek-efek ini bisa membuat drop rate fps dan permainan melambat (*lag*), karena membutuhkan komputasi yang cukup berat, terutama pada perangkat bergerak.

3.1.5. Skenario kasus penggunaan

Pada sub bab ini akan dijelaskan mengenai skenario kasus penggunaan yang terdapat pada sistem. Terdapat beberapa kasus penggunaan. Diagram kasus penggunaan dapat dilihat pada Gambar 3.3. Penjelasan dari masing-masing kasus dapat dilihat pada Tabel 3.2.



Gambar 3.3 Diagram Kasus Penggunaan pada *Food Fest*

Tabel 3.2 Keterangan Kasus Penggunaan pada Food Fest

No	Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
1	UC-P01	Mengadakan festival makanan	Pemain dapat mengadakan festival yang dapat dimainkan bersama dengan pemain lain
2	UC-P02	Mengikuti festival makanan	Pemain dapat mengikuti festival yang sudah tersedia
3	UC-P03	Menampilkan data festival aktif	Daftar festival yang aktif di basis data *server*, akan di kirimkan ke masing-masing *client*
4	UC-P05	Memperbarui data ke *server* pada saat festival berlangsung	Pada saat permainan dimulai, pemain akan senantiasa mengupdate data ke *server*
5	UC-P06	Mengolah data dari *server* pada saat festival berlangsung	*Server* akan mengirimkan data semua pemain ke masing-masing pemain yang tergabung dalam satu festival

Berikut spesifikasi pada masing-masing kasus penggunaan yang terdapat pada *mini game* Food Fest.

3.1.5.1. Mengadakan Festival Makanan

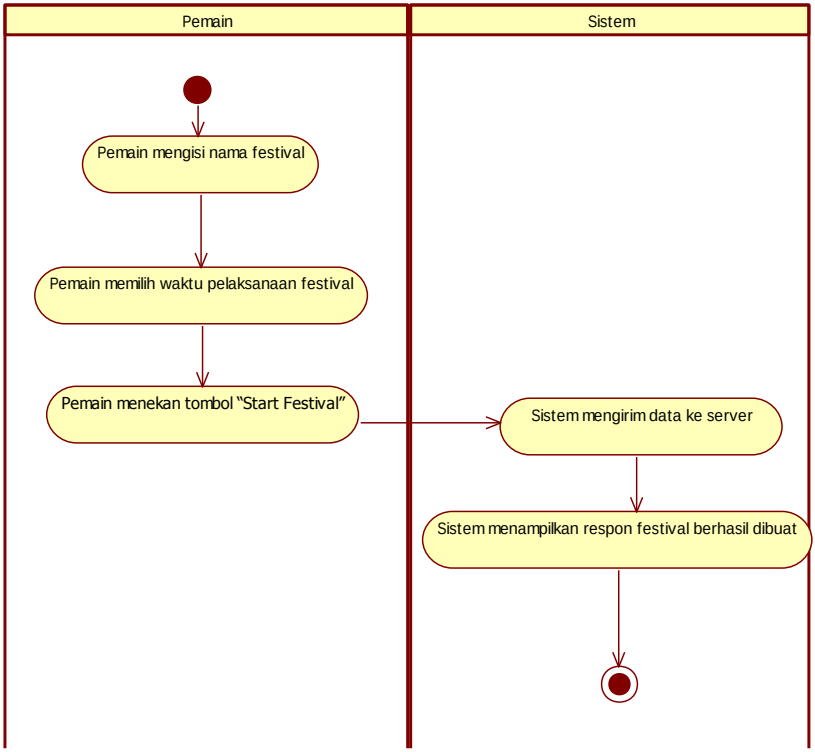
Kasus penggunaan mengadakan festival makanan dilakukan ketika pemain ingin membuat festival makanan yang baru. Pemain memasukkan data yang diperlukan, kemudian data tersebut akan dikirim ke *server* untuk disimpan di dalam basisdata. Spesifikasi kasus penggunaan mengadakan festival makanan dijelaskan pada Tabel 3.3.

Tabel 3.3 Spesifikasi Kasus Penggunaan Mengadakan Festival Makanan

Nama Kasus Penggunaan	Mengadakan Festival Makanan
Kode	UC-001
Deskripsi	Aplikasi pengguna untuk membuat sebuah festival makanan yang baru
Aktor	pemain
Kondisi Awal	Pemain berada pada halaman *create new festival*, pemain terhubung pada jaringan internet
Alur Normal	1. Pemain mengisi nama festival A1 Pemain tidak mengisi *textbox* nama festival 2. Pemain memilih waktu pelaksanaan festival A2 Pemain memilih waktu yang sudah lewat 3. Pemain menekan tombol "Start Festival" 4. Sistem mengirim data ke *server* 5. Sistem menampilkan respon festival berhasil dibuat 6. selesai
Alur Alternatif	A1 Pengguna tidak mengisi *textbox* nama festival A1.1 sistem memberi peringatan untuk mengisi nama festival dengan benar A1.2 menuju alur nomor 1 A2 Pemain memilih waktu yang sudah lewat

	A2.1 sistem memberi peringatan untuk mengisi data waktu dengan benar A1.2 menuju alur nomor 2
Kondisi Akhir	Data tentang festival yang dibuat sudah tersimpan didalam basisdata *server*

Berdasarkan pada spesifikasi kasus penggunaan pada Tabel 3.3 yang melibatkan aktor dan sistem, maka dapat digambarkan diagram aktivitas untuk kasus penggunaan mengadakan festival makanan seperti pada Gambar 3.4.



Gambar 3.4 Diagram Aktivitas Mengadakan Festival Makanan

3.1.5.2. Mengikuti Festival Makanan

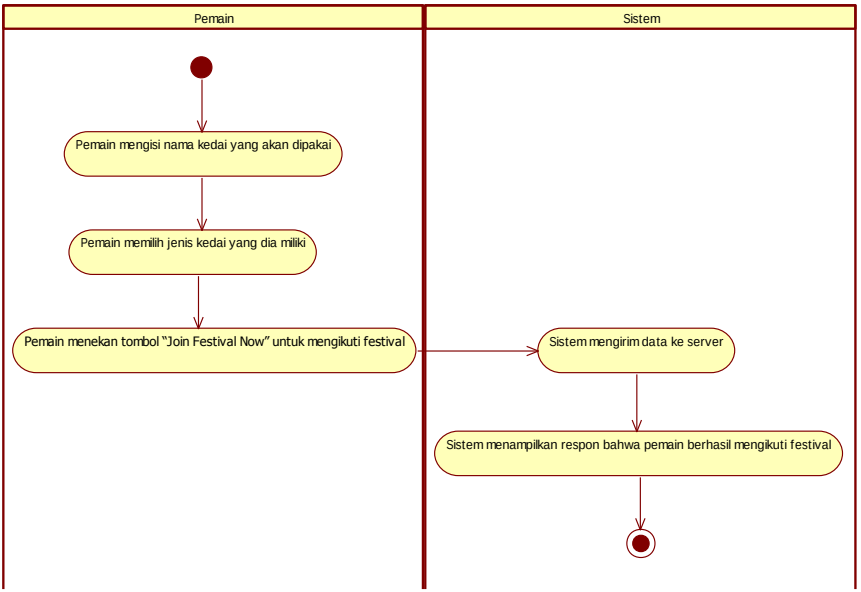
Kasus penggunaan mengikuti festival makanan dilakukan ketika pemain ingin mengikuti festival makanan yang sudah ada maupun festival makanan yang baru saja dia adakan. Pemain memasukkan data yang diperlukan, kemudian data tersebut akan dikirim ke *server* untuk disimpan di dalam basisdata. Spesifikasi kasus penggunaan mengadakan festival makanan dijelaskan pada Tabel 3.4.

Tabel 3.4 Spesifikasi Kasus Penggunaan Mengikuti Festival Makanan

Nama Kasus Penggunaan	Mengikuti Festival Makanan
Kode	UC-002
Deskripsi	Aplikasi pengguna untuk mengikuti festival yang ada
Aktor	Pemain
Kondisi Awal	Pemain sudah memilih salah satu dari festival yang aktif, pemain terhubung pada jaringan internet, pemain berada pada halaman *Join festival*
Alur Normal	1. Pemain mengisi nama kedai yang akan dipakai A1 Pemain tidak mengisi *textbox* nama kedai 2. Pemain memilih jenis kedai yang dia miliki 3. Pemain menekan tombol "Join Festival Now" untuk mengikuti festival 4. Sistem mengirim data ke *server* 5. Sistem menampilkan respon bahwa pemain berhasil mengikuti festival 6. Selesai
Alur Alternatif	A1 Pemain tidak mengisi *textbox* nama kedai A1.1 sistem memberi peringatan untuk mengisi nama kedai dengan benar A1.2 menuju alur nomor 1

Kondisi Akhir	Pemain terdaftar dalam festival dan siap untuk menjalankan *gameplay* festival pada saat festival tersebut sudah dimulai

Berdasarkan pada spesifikasi kasus penggunaan pada Tabel 3.4 yang melibatkan aktor dan sistem, maka dapat digambarkan diagram aktivitas untuk kasus penggunaan mengikuti festival makanan seperti pada Gambar 3.5.



Gambar 3.5 Diagram Aktivitas Mengikuti Festival Makanan

3.1.5.3. Menampilkan Data Festival Aktif

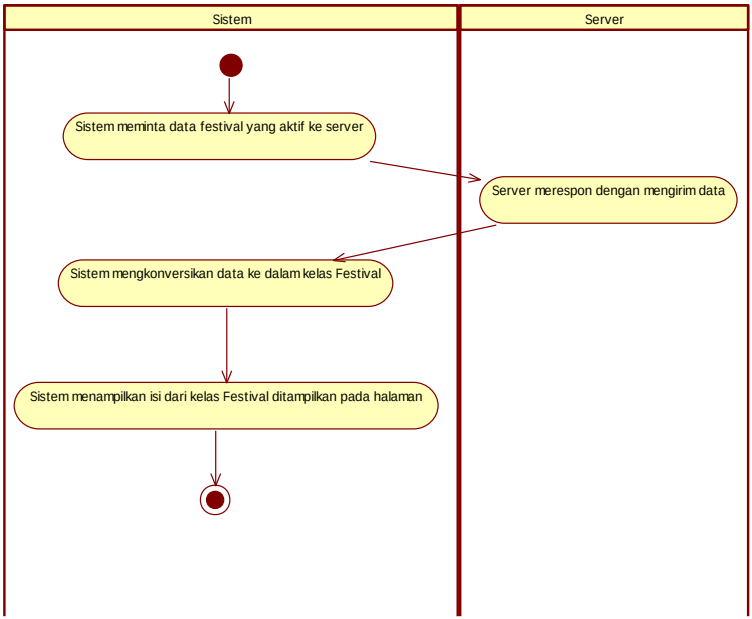
Kasus penggunaan menampilkan data festival aktif dilakukan ketika pemain mengakses halaman yang menampilkan daftar festival. Seketika itu system akan meminta data ke *server*

tentang daftar festival yang aktif. Spesifikasi kasus penggunaan mengadakan festival makanan dijelaskan pada Tabel 3.5.

Tabel 3.5 Spesifikasi Kasus Penggunaan Menampilkan Data Festival Aktif

Nama Kasus Penggunaan	Menampilkan Data Festival Aktif
Kode	UC-003
Deskripsi	Aplikasi sistem untuk menampilkan data festival yang aktif dari basisdata
Aktor	Sistem
Kondisi Awal	Pemain berada pada halaman *festival list*, pemain terhubung pada jaringan internet
Alur Normal	1. Sistem meminta data festival yang aktif ke *server* 2. *Server* merespon dengan mengirim data 3. Sistem mengkonversikan data ke dalam kelas Festival 4. Sistem menampilkan isi dari kelas Festival ditampilkan pada halaman 5. Selesai
Alur Alternatif	-
Kondisi Akhir	Data festival yang aktif tampil di halaman

Berdasarkan pada spesifikasi kasus penggunaan pada Tabel 3.5 yang melibatkan aktor dan sistem, maka dapat digambarkan diagram aktivitas untuk kasus penggunaan menampilkan data festival aktif seperti pada Gambar 3.6.



Gambar 3.6 Diagram Aktivitas Menampilkan Data Festival Aktif

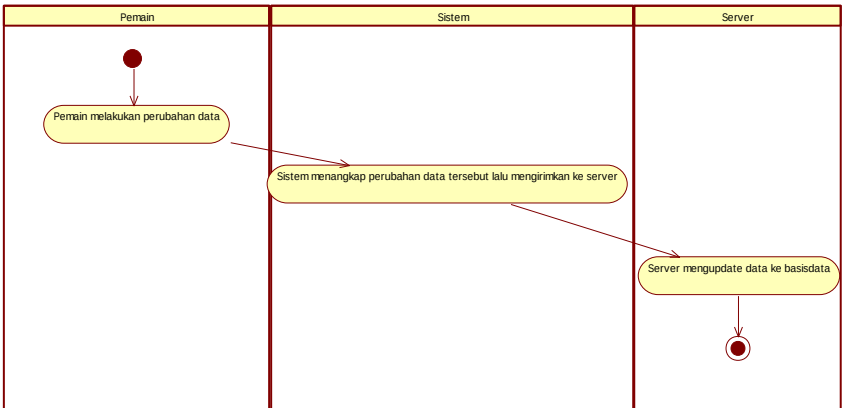
3.1.5.4. Memperbarui Data ke *Server* pada Saat Festival Berlangsung

Kasus penggunaan mengupdate data ke *server* pada saat festival berlangsung dilakukan ketika pemain ingin membuat festival makanan yang baru. Pemain memasukkan data yang diperlukan, kemudian data tersebut akan dikirim ke *server* untuk disimpan di dalam basisdata. Spesifikasi kasus penggunaan mengadakan festival makanan dijelaskan pada Tabel 3.6.

Tabel 3.6 Spesifikasi Kasus Penggunaan Memperbarui Data ke *Server* pada Saat Festival Berlangsung

Nama Kasus Penggunaan	Memperbarui Data ke *Server* pada Saat Festival Berlangsung
Kode	UC-004
Deskripsi	Pada saat festival dimulai, sistem akan secara berkala mengirimkan data ke *server*
Aktor	sistem
Kondisi Awal	Pemain berada pada saat *gameplay* dimulai, pemain terhubung pada jaringan internet
Alur Normal	1. Pemain melakukan perubahan data 2. Sistem menangkap perubahan data tersebut lalu mengirimkan ke *server* 3. *Server* mengupdate data ke basisdata 4. Selesai
Alur Alternatif	-
Kondisi Akhir	Data pemain yang terbaru sudah tersimpan di dalam basis data *server* dan juga semua pemain yang tegabung dalam satu festival mempunyai masing-masing data pemain lainnya yang terbaru.

Berdasarkan pada spesifikasi kasus penggunaan pada Tabel 3.6 yang melibatkan aktor dan sistem, maka dapat digambarkan diagram aktivitas untuk kasus penggunaan memperbarui data ke *server* pada saat festival berlangsung seperti pada Gambar 3.7.



Gambar 3.7 Diagram Aktivitas Mengupdate Data ke *Server* pada Saat Festival Berlangsung

3.1.5.5. Mengolah Data dari *Server* pada Saat Festival Berlangsung

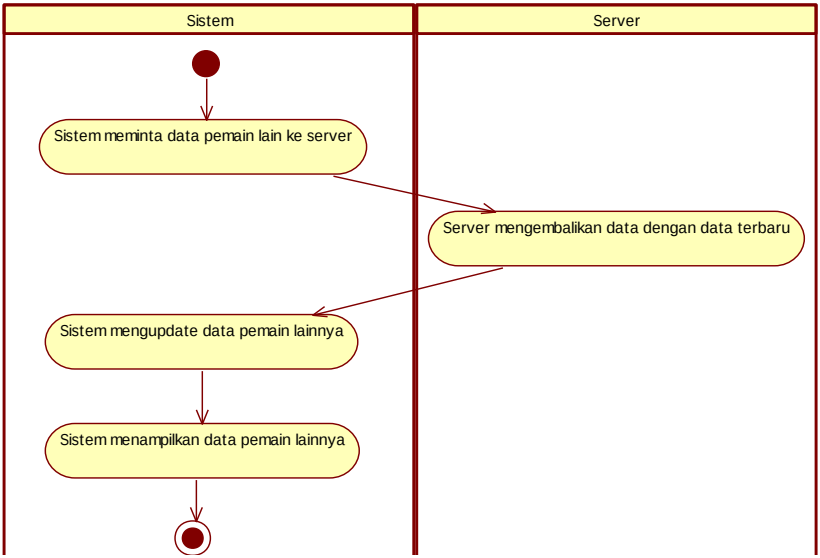
Kasus penggunaan mengolah data dari *server* pada saat festival berlangsung dilakukan ketika *server* mengirimkan data ke masing-masing *client* yang terhubung dalam satu festival. Spesifikasi kasus penggunaan mengadakan festival makanan dijelaskan pada Tabel 3.7.

Tabel 3.7 Spesifikasi Kasus Penggunaan Mengolah Data dari *Server* pada Saat Festival Berlangsung

Nama Kasus Penggunaan	Mengolah Data dari *Server* Pada Saat Festival Berlangsung
Kode	UC-005
Deskripsi	Pada saat festival berlangsung, system akan selalu meminta data terbaru dari pemain lainnya yang tergabung dalam satu festival. Jika ada perubahan maka akan diupdate dan ditampilkan dalam festival
Aktor	System

Kondisi Awal	-
Alur Normal	1. Sistem meminta data pemain lain ke *server* 2. *Server* mengembalikan data dengan data terbaru 3. Sistem mengupdate data pemain lainnya 4. Sistem menampilkan data pemain lainnya 5. Selesai
Alur Alternatif	-
Kondisi Akhir	Data pemain lainnya yang tergabung dalam satu festival terbarui secara berkala

Berdasarkan pada spesifikasi kasus penggunaan pada Tabel 3.7 yang melibatkan aktor dan sistem, maka dapat digambarkan diagram aktivitas untuk kasus penggunaan mengupdate data ke *server* pada saat festival berlangsung seperti pada Gambar 3.8.



Gambar 3.8 Diagram Aktivitas Mengolah Data dari *Server* pada Saat Festival Berlangsung

3.2. Tahap Perancangan

Subbab ini membahas perancangan dari *mini synchronous game* Food Fest. Subbab ini terdiri dari perancangan antarmuka pengguna, dan perancangan permainan.

3.2.1. Perancangan Antarmuka Pengguna

Dalam aplikasi ini terdapat lima tampilan, yaitu tampilan halaman awal, tampilan daftar festival yang aktif, tampilan membuat *gameplay*.

Rancangan tampilan halaman awal seperti yang ditunjukkan pada Gambar 3.9, merupakan halaman pertama yang muncul saat pemain memasuki *mini game*. Pada tampilan ini terdapat judul dari aplikasi, penjelasan tentang Food Fest, tombol untuk masuk ke Food Fest, dan tombol untuk kembali ke *game* utama.

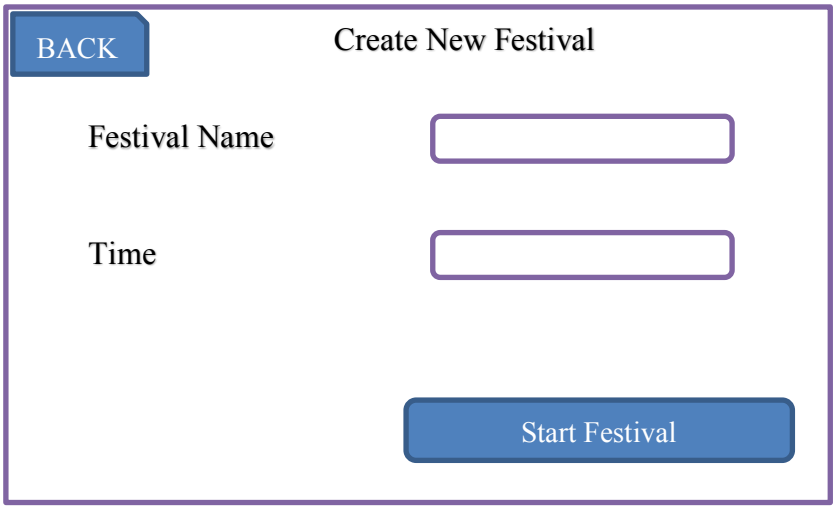
Gambar 3.9 Rancangan Tampilan Awal *Mini Game*

Setelah pemain menekan tombol "Play *Mini Game* Food Fest", maka pemain akan memasuki halaman *festival list* seperti

yang ditunjukkan pada Gambar 3.10. Pada halaman tersebut akan ditampilkan daftar festival yang aktif yang dapat diikuti oleh pemain. Jika pemain ingin membuat festival sendiri, maka pemain dapat menekan tombol "Create New Festival" yang terdapat di pojok kanan bawah.

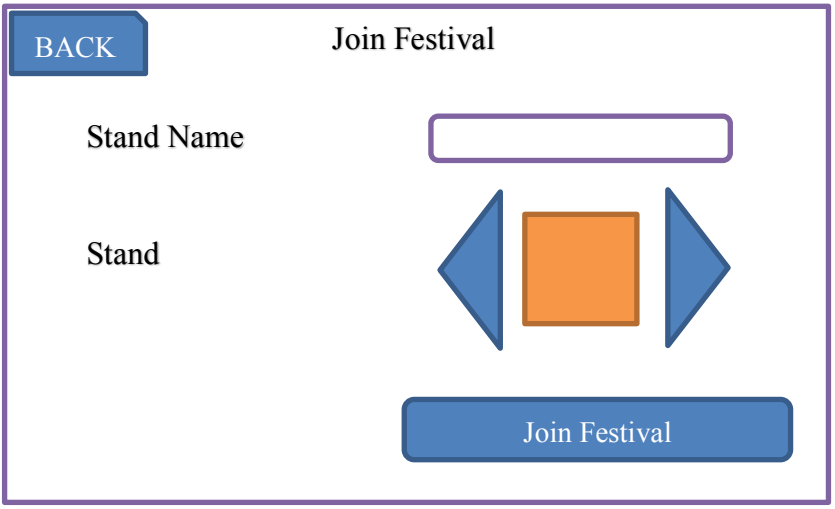
Gambar 3.10 Rancangan Tampilan Halaman Festival List

Setelah pemain menekan tombol "Create New Festival", maka pemain akan memasuki halaman untuk mengisikan data festival baru yang akan dibuat seperti yang ditunjukkan pada Gambar 3.11. Pada halaman tersebut, pemain harus mengisi nama festival dan waktu pelaksanaan festival tersebut. Setelah mengisi data festival, pemain harus menekan tombol "Start Festival" agar data festival yang ingin dibuat bisa disimpan ke dalam basisdata *server*. Setelah itu, pemain akan langsung otomatis menuju halaman join festival untuk mengisi data yang diperlukan untuk mengikuti festival.



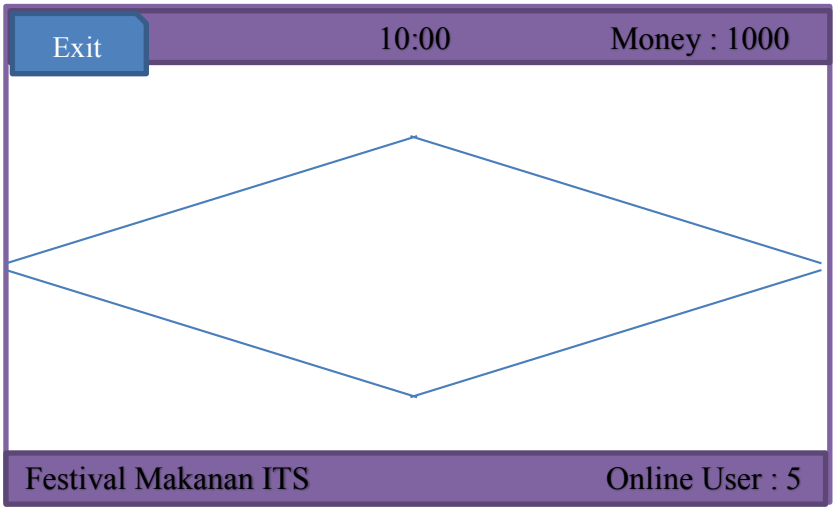
Gambar 3.11 Rancangan Tampilan Halaman Create New Festival

Ketika pemain menekan tombol "Join" pada daftar festival yang ada, maka pemain juga akan masuk ke halaman "Join Festival" untuk mengisi data yang diperlukan untuk festival seperti yang ditunjukkan pada Gambar 3.12. Pemain harus memasukkan nama kedai yang ingin ditampilkan pada saat festival berlangsung. Kemudian pemain harus memilih kedai yang dia punya untuk mengikuti festival. Setelah selesai mengisi data, maka pemain menekan tombol "Join festival". Kemudian data tersebut akan disimpan di dalam basisdata *server*. Setelah itu pemain akan menuju halaman awal dari *mini game*.



Gambar 3.12 Rancangan Tampilan Halaman Join Festival

Pemain yang sudah mengikuti satu festival tidak dapat mengikuti festival lainnya sebelum festival yang diikuti selesai. *Server* akan senantiasa mengecek waktu pelaksanaan festival. Jika festival sudah dimulai, maka *server* akan memberikan sinyal kepada pemain yang tergabung. Pada saat menekan tombol "Play *Mini Game* Food Fest" pada halaman awal, maka pemain akan langsung diarahkan pada tampilan *gameplay* seperti yang ditunjukkan pada Gambar 3.13.



Gambar 3.13 Rancangan Tampilan *Gameplay*

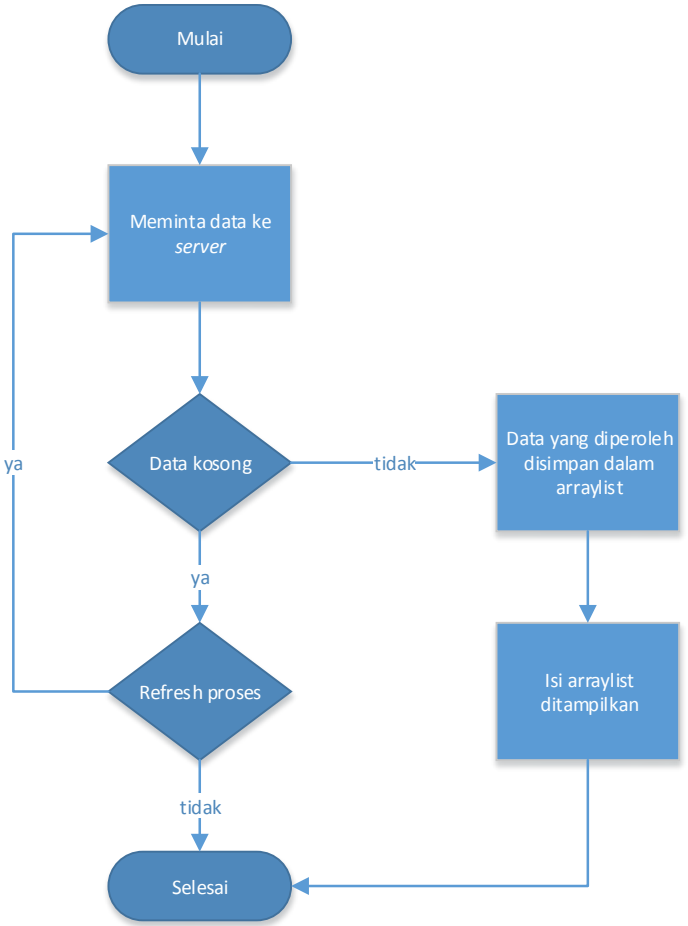
3.2.2. Perancangan Proses

Subbab ini menjelaskan tentang perancangan proses-proses yang ada pada perangkat lunak. Berikut ini adalah rancangan dari proses-proses pada perangkat lunak yang dikembangkan dalam TA ini.

3.2.2.1. Proses Menampilakan Daftar Festival

Proses untuk menampilkan daftar festival aktif dimulai dengan sistem meminta data festival kepada *server*. *Server* kemudian melakukan *query* data ke basisdata *server*, lalu mengembalikan data berupa *string response* kepada sistem. Sistem yang pemperoleh data tersebut kemudian mengolah data untuk menjadi *object* Festival dan disimpan dalam sebuat *arraylist* dari kelas festival. Isi dari *arraylist* tersebut kemudian ditampilkan dengan *method* OnGUI pada halaman daftar festival.

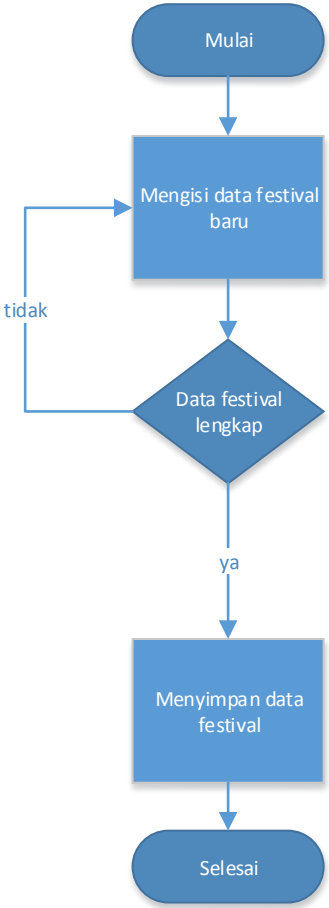
Proses menampilkan daftar festival digambarkan dengan diagram alir seperti yang terlihat pada Gambar 3.14.



Gambar 3.14 Diagram Alir Menampilkan Daftar Festival

3.2.2.2. Proses Membuat Festival Baru

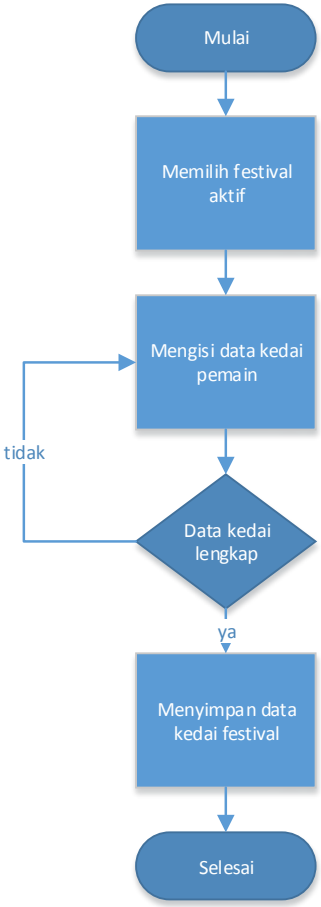
Proses untuk membuat festival baru dimulai dengan pemain mengisi data festival. Kemudian sistem akan mengirim data ke *server* untuk disimpan dalam basisdata *server*. Proses membuat festival baru digambarkan dengan diagram alir seperti yang terlihat pada Gambar 3.15.



Gambar 3.15 Diagram Alir Membuat Festival Baru

3.2.2.3. Proses Mengikuti Festival

Proses untuk mengikuti festival dimulai dengan pemain memilih festival yang aktif. Lalu pemain harus mengisi data kedai yang akan diikutkan. Kemudian sistem akan mengirim data ke *server* untuk disimpan dalam basisdata *server*. Proses mengikuti festival digambarkan dengan diagram alir seperti yang terlihat pada Gambar 3.16.



Gambar 3.16 Diagram Alir Mengikuti Festival

3.2.3. Perancangan Permainan

Pada subbab ini dibahas mengenai rancangan alur permainan dari *mini synchronous game* Food Fest dan karakteristik NPC (*Non-Playable Character*) selaku pembeli.

3.2.3.1. Alur Permainan

Pada dasarnya *gameplay mini game* Food Fest hampir sama dengan *gameplay* utama pada Food Merchant Saga, namun Food Fest dapat dimainkan bersama dalam satu festival. Selain itu Food Fest mempunyai fitur yang lebih sederhana. Berikut elemen-elemen yang ada pada *mini game* Food Fest.

1. Pemain

   Dalam *mini game* Food Fest, pemain bertugas sebagai penjual makanan di dalam sebuah festival makanan. Pemain akan berjual produk makanan yang sudah mereka siapkan dengan pemain lainnya. Pemain dapat melihat kedai pemain lain berserta dengan keuntungan yang sudah didapat. Selain itu, pemain dapat berkomunikasi dengan pemain lain yang tergabung dalam satu festival pada fitur "Player Status". Pada fitur tersebut, pemain dapat menuliskan kata-kata apapun sebagai bentuk ekspresi atau pesan dari pemain. Kemudian kata-kata tersebut dapat dilihat oleh semua pemain lain dalam satu festival.
2. Kedai dan makanan

   Pemain mempunyai bermacam-macam kedai dengan berbagai level makanan. Namun pada Food Fest ini, pemain hanya dapat menampilkan salah satu kedai miliknya dengan level makanan terendah. Sehingga untuk mendapatkan keuntungan yang sebesar-besarnya, pemain harus mempersiapkan makanan sebanyak mungkin untuk memenuhi kebutuhan pembeli.
3. Uang

   Jumlah uang yang dibayar pembeli bergantung pada harga dari makanan yang dibeli. Setelah pembeli membeli produk makanan, animasi uang akan muncul pada layar. Tugas

pemain adalah menekan uang tersebut agar tersimpan dalam data keuntungan pemain. Uang yang terkumpul di akhir festival akan dijumlahkan dengan uang yang pemain miliki.

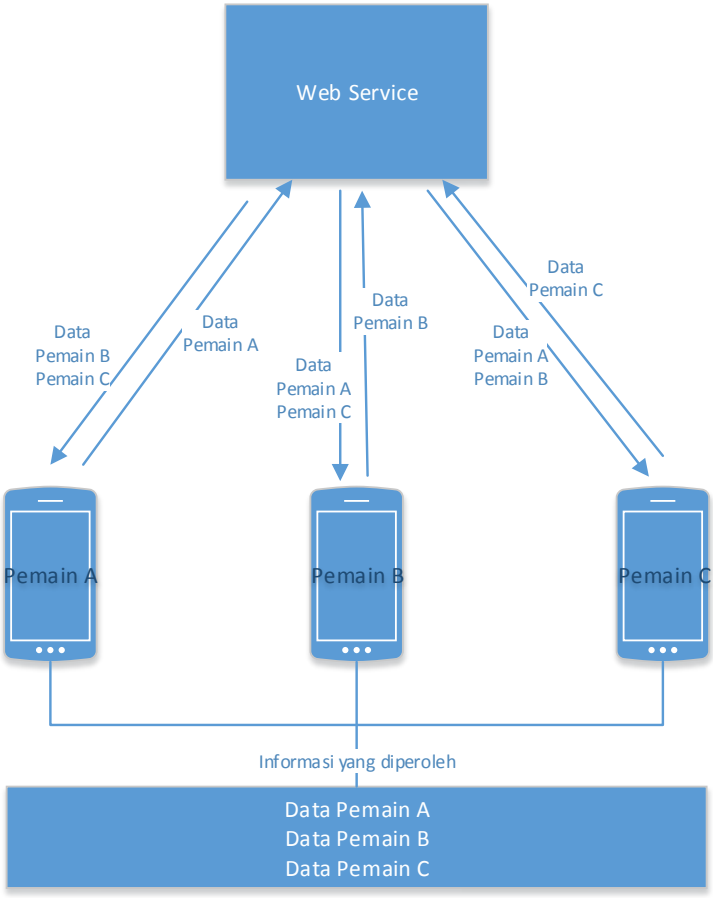
4. Pembeli
   Sama dengan *gameplay* utama pada Food Merchant Saga, pembeli dalam festival dijalankan oleh NPC (*Non-Playable Character*) dengan menerapkan algoritma AI (*Artificial Intelegence*). NPC tersebut dijalankan pada sisi *client.* Sehingga pembeli antara pemain satu dengan pemain lainnya di dalam satu festival yang sama, berbeda. Namun data antar pemain dalam satu festival, sama. Karena setiap hasil interaksi pemain dengan pembeli akan di*update* ke *server*. Lalu *server* akan menyebar informasi pada pemain lainnya.
5. Waktu permainan
   Festival akan berlangsung sesuai dengan jadwal yang dituliskan oleh pembuat festival. Kemudian lama festival berlangsung adalah 15 menit. Pemain tidak dapat bergabung setelah festival dimulai. Pemain yang sudah tergabung namun tidak mengakses halaman *gameplay*, akan terlihat pasif dan tidak ada perubahan data yang terjadi.

3.2.3.2. Alur Sinkronisasi

Data antar pemain disimpan pada basisdata *server* menggunakan *web service.* Data tersebut juga dikembalikan ke pemain lainnya sebagai suatu informasi. Proses keluar masuknya data tersebut adalah alur sinkronisasi yang terjadi pada saat festival berlangsung. Diagram mengenai alur sinkronisasi dapat dilihat pada Gambar 3.17.

Gambar 3.17 menggambarkan alur sinkronisasi pada saat festival berlangsung. Pada gambar tersebut terdapat 3 pemain, dimana ketiga pemain tersebut akan saling bertukar data. Pemain A memberikan data pemain A pada *web service*. Kemudian pemain A mendapatkan data pemain yang lainnya, yaitu pemain B dan pemain C. Begitu juga dengan pemain B dan pemain C,

keduanya akan mendapatkan data pemain yang lainnya. Sehingga informasi yang didapat ketiga pemain tersebut adalah sama, yaitu data pemain A, pemain B, dan pemain C.



Gambar 3.17 Diagram Alur Sinkronisasi

Data yang terdapat pada pemain akan saling bertukar dengan *server* dan pemain lainnya. Data tersebut terdiri dari

Idfestival, uang, kedai festival, dan status pemain. Berikut adalah elemen sinkronisasi data.

1. Id Transaksi kedai
   Merupakan id atau kode khusus dimana pemain membeli sebuah kedai. Dari data id ini, kedai pemain dapat dimunculkan pada saat festival berlangsung.
2. Nama Kedai festival
   Merupakan nama kedai yang didaftarkan oleh pemain.
3. Id Festival
   Merupakan id atau kode khusus pada festival yang diikuti oleh pemain.
4. Uang pemain
   Merupakan uang yang dimiliki oleh pemain. Uang ini pada akhirnya akan diakumulasi dengan total uang pemain pada permainan utama.
5. Status pemain
   Merupakan fitur tambahan dimana pemain bisa berkirim pesan atau berbagi status dengan pemain lainnya pada saat festival berlangsung.

3.2.3.3. Karakteristik NPC (*Non-Playable Character*)

NPC menjalankan peran sebagai pembeli. Pembeli akan berjalan menyusuri ruangan festival dengan menerapkan sebuah algoritma kecerdasan buatan. NPC dijalankan dengan menggunakan dua algoritma yaitu behaviour trees dan A*.

Begaviour trees digunakan untuk menentukan tindakan NPC sesuai dengan lingkungan. Dia bisa beli sesuatu ataukah hanya berjalan-jalan saja. Kemudian algoritma A* digunakan untuk proses penemuan jalan yang akan dipilih oleh NPC atau biasa disebut dengan sebutan *path finding*.

Karakteristik NPC ini sudah diterapkan pada modul AI. *Mini game* perlu mengintegrasikan modul AI untuk bisa membangkitkan karakteristik NPC pada *gameplay*.

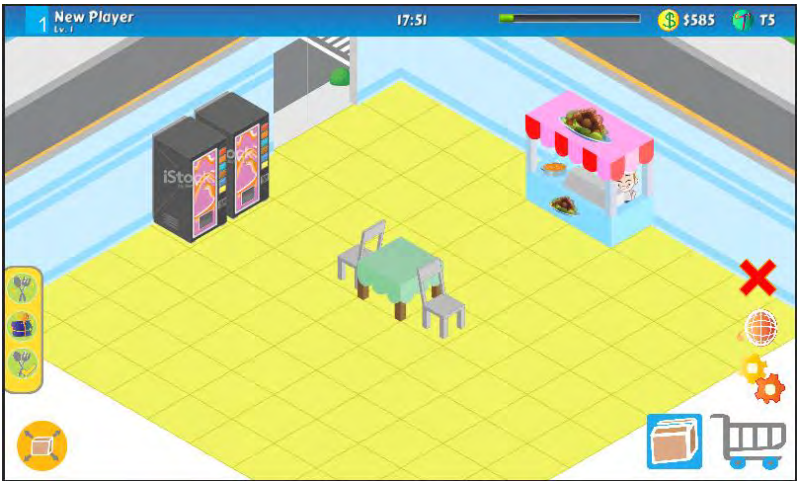
BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman C#.

4.1. Implementasi Antarmuka

Pada subbab ini akan dijelaskan mengenai implementasi dari antarmuka yang akan dilihat oleh pemain. Terdapat lima tampilan antarmuka yang dibuat. Kesemuanya diatur dengan posisi *panel*, *button*, *textbox* dengan bantuan dari *plugin* NGUI. Plugin ini dapat diunduh secara gratis di Unity Asset Store.

Pada saat pemain menjalankan *game* Food Merchant Saga, pemain akan langsung diarahkan ke *gameplay* utama. *Game* play tersebut berisikan Pujasera dan kedai yang dimiliki oleh pemain. Gambar mengenai *gameplay* pada Food Merchant Saga dapat dilihat pada Gambar 4.1.



Gambar 4.1 Tampilan *Gameplay* pada Food Merchant Saga

Pada gambar tersebut, terdapat menu online yang terletak didaerah sekitar kanan. Menu tersebut digunakan untuk memberi hadiah kepada teman, mencari teman, dan juga mengakses Food Fest melalui tombol Festival yang telah disediakan. Ketika pemain menekan tombol festival, maka pemain akan diarahkan ke *scene* baru yang menangani tentang fitur festival pada Food Merchant Saga. Terdapat lima halaman antarmuka pada Food Fest, yaitu halaman awal, halaman daftar festival, halaman membuat festival, halaman mendaftarkan kedai ke festival, dan halaman *gameplay*.

4.1.1. Halaman Awal

Pada saat pemain mengakses Food Fest, maka akan muncul tampilan halaman awal yang berisi penjelasan Food Fest dan tombol navigasi. Gambar 4.1 menunjukkan tampilan halaman awal yang dibuat. Pada gambar tersebut terdapat dua tombol. Tombol back akan mengarahkan pemain pada *gameplay* utama Food Merchant saga. Kemudian tombol play *mini game* food fest akan mengarahkan pemain ke festival list.

Gambar 4.2 Tampilan Halaman Awal *Game*

Pada saat memasuki halaman awal, maka dijalankan kelas init() yang mengatur kondisi awal aplikasi. Berikut adalah kode sumber inisialisasi *game*.

```
void Init ()
{
	status = 1;
	//button
	btnLoad = GameObject.Find ("Button - Loading");
	btnPlay = GameObject.Find ("Button - Play");
	btnPlayFestivalNow = GameObject.Find ("Button -
Play Festival Now");
	btnCreate = GameObject.Find ("Button -
Create");
	btnStart = GameObject.Find ("Button - Start");
	btnJoin = GameObject.Find ("Button - Join
Now");
	btnPlayFestivalNow.transform.Translate(0,
20,0);

	//page
	mainPage = GameObject.Find ("MainPage");
	festivalListPage = GameObject.Find
("FestivalListPage");
	createFestivalPage = GameObject.Find
("CreateFestivalPage");
	passivePage = GameObject.Find ("PassivePage");
	joinFestivalPage =
GameObject.Find("JoinFestivalPage");

	rx = Screen.width * 55 / 800;
	ry = Screen.height * 90 / 480;
	rw = Screen.width * 17 / 20;
	rh = Screen.height * 13 / 24;

	tbNama = GameObject.Find("UILabel -
name").GetComponent<UILabel>();
	tbD = GameObject.Find("UILabel -
day").GetComponent<UILabel>();
```

```
        tbM = GameObject.Find("UILabel -
month").GetComponent<UILabel>();
        tbY = GameObject.Find("UILabel -
year").GetComponent<UILabel>();
        tbTimes = GameObject.Find("UILabel -
times").GetComponent<UILabel>();
        tbStandName = GameObject.Find("UILabel - stand
name").GetComponent<UILabel>();
        SetDateTime ();
}
```

Kode Sumber 4.1 Inisialisasi *Game* pada Fungsi `Init()`

Pada Kode Sumber 4.1 tersebut terlihat bahwa semua *gameobject* dijadikan suatu variabel. Variable tersebut akan diolah dan digunakan untuk proses navigasi antar halaman. Proses navigasi antar satu halaman ke halaman lain dijalankan dengan cara translasi halaman. Artinya, antarmuka halaman berada dalam satu *scene* hanya saja fokus kamera mengarah pada halaman-halaman tertentu yang ingin dituju.

4.1.2. Halaman Tampilan Daftar Festival

Setelah pemain menekan tombol "Play *Mini Game* Food Fest", maka pemain akan langsung diarahkan pada halaman festival list. Pada halaman tersebut akan ditampilkan daftar festival yang aktif yang dapat diikuti oleh pemain. Gambar 4.3 menunjukkan tampilan halaman daftar festival aktif.

Untuk dapat menuju halaman tampilan daftar festival, variabel *gameobject* yang merepresentasikan halaman awal ditranslasi ke atas dan variabel *gameobject* yang merepresentasikan halaman daftar festival aktif ditranslasi ke bawah. Hal tersebut ditunjukkan pada Kode Sumber 4.2 dan Kode Sumber 4.3.

Untuk menampilkan daftar festival, sistem akan meminta data dengan memanggil *method* "loadFestivalList". *Method* ini akan dijelaskan pada subbab implementasi proses. Kemudian, data yang didapat disimpan didalam variable respon. Dari

variable response, diolah untuk dijadikan sebuah obyek Festival kemudian disimpan dalam suatu arraylist. Dari arraylist tersebut, data kemudian dipanggi oleh fungsi `OnGUI` untuk ditampilkan dalam sebuah scroll bar. Kode Sumber 4.4 menunjukkan bagaimanan fungsi `OnGUI` menampilkan daftar festival.

Gambar 4.3 Tampilan Daftar Festival Aktif

```
void OnClickPlay (GameObject go)
{
        status = 2;
        MoveFromMainPage ();
}
```

Kode Sumber 4.2 Fungsi `OnClickPlay`

```
void MoveFromMainPage ()
{
        if (status == 1)
        {
        Application.LoadLevel("GamePlayTest");
        } else if (status == 2)
```

```
{
              mainPage.transform.Translate (0, 20, 0);
              btnPlayFestivalNow.transform.Translate
(0, -20, 0);
              festivalListPage.transform.Translate (-
20, 0, 0);
              StartCoroutine("ILoadFestivalList");
       }
}
```

Kode Sumber 4.3 Fungsi `MoveFromMainPage()`

```
void OnGUI()
{
       if (status == 2)
       {
              string namaFest;
              GUILayout.BeginArea (new Rect (rx, ry,
rw, rh));
              scrollPosition =
GUILayout.BeginScrollView(scrollPosition,
GUILayout.Width(rw), GUILayout.Height(rh));

       if(listFestival!=null)
              for(int i=0; i<listFestival.Count; i++)
              {
                     FFFestival fests;
                     fests = listFestival[i] as
FFFestival;
                     namaFest =
fests.namaFestival.Replace("_", " ");
                     GUILayout.Label("Festival Id : " +
fests.idFestival);
                     GUILayout.Label("Festival Name : "
+ namaFest);
                     GUILayout.Label("Date Time
: " + fests.tanggalMulai);

                     if(GUILayout.Button("Join
Festival") && statusIkutFestival==false)
```

```
                    {
                        idFest=fests.idFestival;
                        status=4;
                    MoveFromFestivalListPage();
                    }
            }
            GUILayout.EndScrollView();
            GUILayout.EndArea ();
        }
}
```

Kode Sumber 4.4 Fungsi `OnGUI()` yang Menampilkan Daftar Festival Aktif

4.1.3. Halaman Membuat Festival Baru

Pemain dapat membuat festival baru dengan menekan tombol "Create New Festival" pada halaman daftar festival. Pada halaman tersebut, pemain harus mengisi nama festival dan waktu pelaksanaan festival tersebut. Setelah mengisi data festival, pemain harus menekan tombol start festival agar data festival yang ingin dibuat bisa disimpan ke dalam basis data *server*. Setelah itu, pemain akan langsung otomatis menuju halaman join festival untuk mengisi data yang diperlukan untuk mengikuti festival. Gambar 4.4 menunjukkan tampilan halaman create new festival.

Untuk dapat menuju halaman membuat festival baru, variabel *gameobject* yang merepresentasikan halaman daftar festival ditranslasi ke atas dan variabel *gameobject* yang merepresentasikan halaman membuat festival baru ditranslasi ke bawah. Hal tersebut ditunjukkan pada Kode Sumber 4.5 dan Kode Sumber 4.6.

Data yang dimasukkan oleh pemain akan disimpan terlebih dahulu pada sebuah variable. Pada saat pemain menekan tombol start, maka sistem akan mengirimkan data ke *server* untuk disimpan dalam basisdata *server*.

Gambar 4.4 Tampilan Halaman Membuat Festival Baru

```
void OnClickCreate (GameObject go){
	if (statusIkutFestival == false) {status = 3;
		MoveFromFestivalListPage ();
	}
}
```

Kode Sumber 4.5 Fungsi `OnClickCreate`

```
void MoveFromFestivalListPage (){
	if (status == 2) {
		mainPage.transform.Translate (0, -20, 0);
		btnPlayFestivalNow.transform.Translate
(0, 20, 0);
festivalListPage.transform.Translate (20, 0, 0);
		status = 1;
		LoadDataFestival ();
	} else if (status == 3) {
festivalListPage.transform.Translate (20, 0, 0);
createFestivalPage.transform.Translate (20, 0, 0);
	} else if (status == 4) {
```

```
                festivalListPage.transform.Translate (20,
0, 0);
                joinFestivalPage.transform.Translate (0,
20, 0);
                foreach(Stall o in listStall)
                {
        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, -13, 0);
                }
        }
}
```

Kode Sumber 4.6 Fungsi `MoveFromFestivalList()`

4.1.4. Halaman Mengikuti Festival

Setelah pemain membuat festival baru, maka pemain akan langsung menuju halaman form mengikuti festival. Selain itu, pemain yang menekan tombol "Join" pada halaman daftar festival juga akan mengarah pada halaman ini. Gambar 4.5 menunjukkan tampilan halaman mengikuti festival.

Gambar 4.5 Tampilan Halaman Mengikuti Festival.

Pada gambar tersebut terlihat bahwa pemain harus memasukkan nama kedai yang ingin ditampilkan pada saat festival berlangsung. Kemudian pemain juga harus memilih kedai yang akan dipakai. Setelah itu pemain menekan tombol join untuk memasukkan data ke *server*.

Untuk dapat menuju halaman membuat mengikuti festival, terdapat dua cara tergantung darimana pemain dapat mengakses halaman mengikuti festival. Pertama, jika pemain sudah selesai membuat festival baru, maka pemain akan langsung menuju halaman mengikuti festival. Pada cara ini, variabel *gameobject* yang merepresentasikan halaman membuat festival baru ditranslasi ke atas dan variabel *gameobject* yang merepresentasikan halaman mengikuti ditranslasi ke bawah. Hal tersebut ditunjukkan pada Kode Sumber 4.7.

```
void OnClickJoin (GameObject go){
       status = 1;
       btnStart.transform.Translate (0, 20, 0);
       btnJoin.transform.Translate (20, 0, 0);
       btnLoad.transform.Translate (0, -20, 0);
       if (!waitActive) {
              StartCoroutine (Wait ());
       }
}
```

Kode Sumber 4.7 Menuju Halaman Mengikuti Festival dari Halaman Membuat Festival Baru dengan Fungsi `OnClickJoin`

Kemudian cara kedua adalah pemain mengakses halaman mengikuti festival dari halaman daftar festival. Disini pemain menekan tombol "Join" yang ada pada masing-masing festival. Pada cara ini, variabel *gameobject* yang merepresentasikan halaman daftar festival ditranslasi ke atas dan variabel *gameobject* yang merepresentasikan halaman mengikuti ditranslasi ke bawah. Hal tersebut ditunjukkan pada Kode Sumber 4.8 dan Kode Sumber 4.9.

```
if(GUILayout.Button("Join Festival") &&
statusIkutFestival==false){
        idFest=fests.idFestival;
        status=4;
        MoveFromFestivalListPage();
}
```

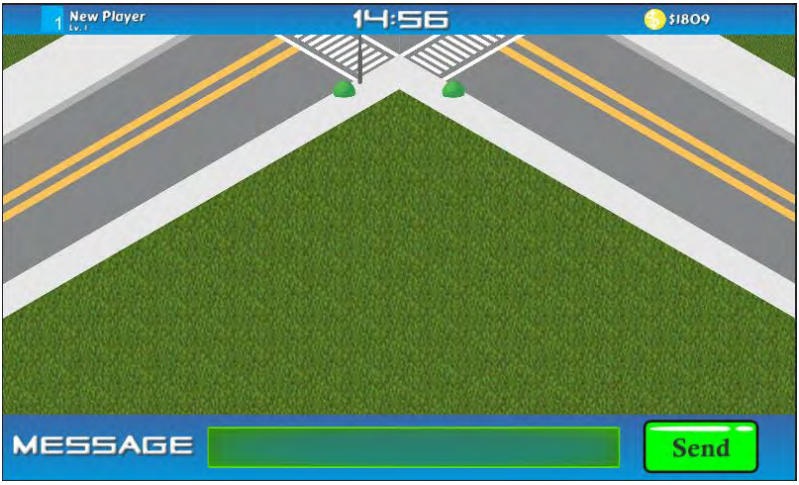
Kode Sumber 4.8 Menuju Halaman Mengikuti Festival dari Halaman Daftar Festival

```
void MoveFromFestivalListPage (){
        if (status == 2) {
mainPage.transform.Translate (0, -20, 0);
btnPlayFestivalNow.transform.Translate (0, 20, 0);
festivalListPage.transform.Translate (20, 0, 0);
                status = 1;
                LoadDataFestival ();
        } else if (status == 3) {
festivalListPage.transform.Translate (20, 0, 0);
                createFestivalPage.transform.Translate
(20, 0, 0);
        } else if (status == 4) {
                festivalListPage.transform.Translate (20,
0, 0);
                joinFestivalPage.transform.Translate (0,
20, 0);
                foreach(Stall o in listStall)
                {
        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, -13, 0);
                }
        }
}
```

Kode Sumber 4.9 Fungsi `MoveFromFestivalListPage()`

4.1.5. Implementasi *Screen Gameplay*

Pada saat festival sudah dimulai, maka *server* akan memberikan sinyal kepada pemain yang tergabung. Pada saat menekan tombol "Play *Mini Game* Food Fest" pada halaman awal, maka pemain akan langsung diarahkan pada tampilan *gameplay* seperti yang ditunjukkan pada Gambar 4.6.



Gambar 4.6 Tampilan Lokasi Festival

Aktivitas jual beli festival berlangsung pada daerah tempat tile yang digenerate berada. Kode Sumber 4.10 menunjukkan bagaimana tile degenerate dengan memanggil fungsi TileInstantiate pada saat *gameplay* diakses.

```
void TileInstantiate ()
{
        arrayX = new double[100,100];
        arrayY = new double[100,100];

        for(int i = 0; i<100; i++)
        {
                for(int j = 0; j<100; j++)
```

```
                {
                        arrayX[i,j] = (0.89 * j) - (0.89 *
i);
                        arrayY[i,j] = 3 - (0.515 * j) -
(0.515 * i);
                }
        }

        curTile = 20;
        for (int i = 0; i<curTile; i++)
        {
                for (int j = 0; j<curTile; j++)
                {
                        var tileInstant =
Instantiate(tilePrefab) as Transform;
                        tileInstant.position = new
Vector3((float)arrayX[i,j], (float)arrayY[i,j],
FFTile.transform.position.z);
                        tileInstant.parent =
FFTile.transform;
                        TileScript tilePos =
tileInstant.GetComponent<TileScript>();
                        tilePos.posX = i;
                        tilePos.posY = j;
                        tilePos.gridPosition = new
GridPosition(i, j);
                }
        }
}
```

Kode Sumber 4.10. Fungsi `TileInstantiate()`

4.2. Implementasi Proses

Bagian ini menjelaskan tentang implementasi proses-proses pada perangkat lunak yang dikembangkan dalam Tugas Akhir ini. Implementasi proses dilakukan berdasarkan pada perancangan proses yang telah dijelaskan pada bab sebelumnya. Berikut ini adalah implementasi dari proses yang terdapat pada Food Fest.

4.2.1. Implementasi Proses Menampilkan Daftar Festival

Pada saat pemain mengakses halaman daftar festival, sistem menjalankan sebuah fungsi *loadFestivalList*. Fungsi ini meminta data melalui *web service* tentang daftar festival yang aktif. *Server* mengembalikan data berupa *string response*. Kemudian sistem akan mengkonversikan data tersebut menjadi sebuah kelas Festival. Kode Sumber 4.11 menunjukkan isi dari kelas `IloadFestivalList()`.

```
IEnumerator ILoadFestivalList ()
{
	WWW www = new WWW(service.LOADFESTIVALLIST);
	float elapsedTime = 0.0f;
	while (!www.isDone) {
		elapsedTime += Time.deltaTime;
		if (elapsedTime >= 10.0f) break;
		yield return null;
	}
	if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
		Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
		yield break;
	}
	this.response = www.text;
	IList search = (IList)
Json.Deserialize(response);
	listFestival = new ArrayList ();
	foreach (IDictionary fest in search)
	{
		festival = new FFFestival();
		//menampung ke festival
		festival.idFestival =
int.Parse((string)fest["id_festival"]);
		festival.namaFestival =
(string)fest["nama_festival"];
		festival.tanggalMulai =
(string)fest["tanggal_mulai"];
```

```
                festival.jumlahPemain =
int.Parse((string)fest["jumlah_pemain"]);
                listFestival.Add(festival);
        }
}
```

Kode Sumber 4.11 Fungsi `IloadFestivalList()`

4.2.2. Implementasi Proses Membuat Festival Baru

Pada saat pemain mengakses halaman membuat festival baru, pemain akan memasukkan data festival yang diperlukan. Kemudian setelah pemain selesai mengisi data, sistem menjalankan sebuah fungsi *addFestival*. Fungsi ini mengirimkan data yang diisi pemain ke *server* untuk disimpan pada basisdata *server*. Kode Sumber 4.12 menunjukkan isi dari kelas `IAddFestival()`.

```
IEnumerator IAddFestival()
{
        string namaFest, dateFest, times;
        string yy, mm, dd;
        namaFest = tbNama.text;
        yy = tbY.text;
        mm = tbM.text;
        dd = tbD.text;
        if(tbTimes.text=="09 AM")
                times = "09_00_00";
        else
                times = "15_00_00";
        namaFest = namaFest.Replace(" ", "_");
        dateFest = string.Concat (yy,"_",mm,"_",dd);
        WWW www = new
WWW(service.ADDFESTIVAL+namaFest+"/"+dateFest+"/"+time
s);
        float elapsedTime = 0.0f;
        while (!www.isDone) {
                elapsedTime += Time.deltaTime;
                if (elapsedTime >= 10.0f) break;
                yield return null;
```

```
        }
        if (!www.isDone || !string.IsNullOrEmpty
(www.error)) {
                Debug.LogError (string.Format ("Fail
Whale!\n{0}", www.error));
                yield break;
                }
        string response = www.text;
        IList search = (IList)
Json.Deserialize(response);
        listFestival = new ArrayList ();
        foreach (IDictionary fest in search)
        {
                this.idFest =
int.Parse((string)fest["message"]);
        }
}
```

Kode Sumber 4.12 Fungsi `IAddFestival()`

4.2.3. Implementasi Proses Mengikuti Festival

Pada saat pemain menekan tombol "Join" pada daftar festival, pemain akan menuju ke halaman mengikuti festival. Disini pemain harus memasukkan data kedai yang akan diikutkan pada festival. Kemudian setelah pemain selesai mengisi data, sistem menjalankan sebuah fungsi *joinFestival*. Fungsi ini mengirimkan data yang diisi pemain ke *server* untuk disimpan pada basisdata *server*. Kode Sumber 4.13 menunjukkan isi dari kelas `JoinFestival()`.

```
IEnumerator JoinFestival() {
        string idPlayer = PlayerData.playerid;
        tbStandName.text = tbStandName.text.Replace ("
", "_");
        WWW www = new
WWW(service.JOINFESTIVAL+tbStandName.text+"/"+idPlayer
+"/"+standStatus+"/"+idFest);
        float elapsedTime = 0.0f;
```

```
        while (!www.isDone) {
                elapsedTime += Time.deltaTime;
                if (elapsedTime >= 10.0f) break;
                yield return null;
        }
        if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
                Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
                yield break;
        }
        string response = www.text;
        SaveDataFestival (idFest, tbStandName.text,
standStatus);
}
```

Kode Sumber 4.13 Fungsi `JoinFestival()`

4.3. Implementasi *Gameplay*

Pada subbab ini akan dijelaskan tentang implementasi pada *gameplay*. Implementasi mencakup fungsionalitas dan penerapan *object* yang ada pada *gameplay*. Berikut adalah implementasi pada *gameplay* Food Fest.

4.3.1. Data Pemain

Pada *game* utama Food Merchant Saga, terdapat data set yang menyimpan data dari pemain. Data pemain tersebut juga disimpan ke dalam sebuah kelas global. Data tersebut disimpan dalam kelas player data seperti dalam Kode Sumber 4.14.

```
namespace Assets.Scripts.FMSData
{
    [System.Serializable]
    public class PlayerData
    {
                public string playerid;
                public bool haslogin;
                public string pujasera;
```

```
        public int curTile;
        public string name;
        public string avatar;
        public int level;
        public int popularity;
        public int money;
        public int token;
        public int xp;
              public int jmlProperty;
              public int jmlAccesories;
              public int jmlTransaksi;
        public int kodeTransaksi;
    }
}
```

Kode Sumber 4.14 Kelas `PlayerData`

Dalam Food Fest, data yang dibutuhkan adalah *playerid* dan *money*. Untuk kedai pemain, diambil dari FMS Object pada panel manager di Food Merchant Saga. Kode Sumber 4.15 menunjukkan data kedai disimpan pada *linked list*.

```
void initKedaiPemain ()
{
       jumlahStand = 0;
       foreach (FMSObject d in
Managers.Data.ListUsedFMS)
       {
              if(d is Stall)
              {
                     Stall a = d as Stall;
                     GameObject o
=(GameObject)Instantiate(Loader.Load(a.PrefabName));
              o.transform.Translate(2+(15*jumlahStand),
10, 0);
                     listStall.Add(a);
                     jumlahStand++;
                     standStatus=1;
              }
```

```
        }
}
```

Kode Sumber 4.15 Fungsi `initKedaiPemain()`

4.3.2. Pertukaran Data pada Festival

Pemain yang tergabung dalam satu festival, akan saling bertukar data. Data tersebut diperoleh dari basisdata *server* yang diakses melalui *web service.* Pertama-tama pemain yang tergabung dalam satu festival meminta data tentang kedai festival yang sudah didaftarkan pemain lainnya. Proses meminta data ini dapat ditunjukkan pada Kode Sumber 4.16.

```
private void loadKedaiFestivalServer ()
{
        WWW www = new
WWW("http://fms.effolabs.com/indeks.php/FM/loadstall
/"+festivalData.idFestival);

        while(!www.isDone)
        {
                //Managers.Debug.print("end");
        }

        if (www.isDone &&
string.IsNullOrEmpty(www.error)) {
                dataUrl = www.text;
                string response = dataUrl;
                listKedaiUrl = (List<object>)
Json.Deserialize(response);
                listKedai = new ArrayList ();
                int a = 1;
                IDictionary ked;

                for(int i=0; i<listKedaiUrl.Count;
i++)
                {
```

```
                        ked =
(IDictionary)listKedaiUrl[i];
                        kedaiPemain = new
FFKedaiPemain();
                        kedaiPemain.namaKedai =
(string)ked["nama_kedai_festival"];
                        kedaiPemain.idKedai =
int.Parse((string)ked["id_kedai"]);
                        kedaiPemain.makanan_tersedia =
int.Parse((string)ked["makanan_tersedia"]);
                        kedaiPemain.prefabKedai =
namaPrefabs(kedaiPemain.idKedai);
                        kedaiPemain.posisiKedaiX =
lokasiX(a);
                        kedaiPemain.posisiKedaiY =
lokasiY(a);

                if(a==1 || a==2)
                        kedaiPemain.statusPosisi=1;
                else if(a==3 || a==4 || a==5)
                        kedaiPemain.statusPosisi=2;
                else if(a==6 || a==7 || a==8)
                        kedaiPemain.statusPosisi=3;
                else if(a==9 || a==10)
                        kedaiPemain.statusPosisi=4;
                        listKedai.Add (kedaiPemain);
                        a++;
                }
                loadKedaiFestival ();
        }
}
```

Kode Sumber 4.16 Fungsi `loadKedaiFestivalServer()`

Data yang didapat disimpan dalam sebuah kelas KedaiPemain. Setelah pemain mendapatkan data tersebut, maka sistem akan membuat *prefab* kedai pemain untuk ditampilkan pada *gameplay*. Gambar 4.7.

Gambar 4.7 Kedai Festival

Data kedai pemain pada penyimpanan *client* dengan data kedai pemain pada basisdata *server* akan saling bertukar dan kemudian saling bersinkronisasi dengan data pemain lainnya. Data tersebut terdiri dari uang pemain, kedai festival, id festival, dan status pemain.

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada *plug-in* yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsionalitas, integritas, dan performa dari sistem. Pengujian fungsionalitas mengacu pada kasus penggunaan pada bab tiga. Pengujian integritas bertujuan untuk menguji apakah modul yang dibuat dapat terintegrasi dengan modul lainnya. Dan pengujian performa dilakukan untuk mengetahui ketahanan sistem terhadap sejumlah pengguna yang berbeda. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan tugas akhir ini dilakukan pada lingkungan dan alat kakas sebagai berikut:

Jenis Device	: Asus X550CC
Prosesor	: Intel Core i5 1.80 GHz
Memori	: 4 GB RAM
Kartu Grafis	: NVidia Geforce 720M
Sistem Operasi	: Windows 8.1

Jenis Device	: Advan Vandroid T3C
Prosesor	: Cortex A7 Quad Core 1.2 GHz
Memori	: 1.00 GB
Sistem Operasi	: Android 4.2 Jelly Bean

5.2. Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian yang dilakukan adalah pengujian kebutuhan fungsionalitas, pengujian modular dan integritas, dan pengujian transaksi data. Pengujian fungsionalitas menggunakan metode kotak hitam (*black box*). Metode ini menekankan pada

kesesuaian hasil keluaran sistem secara modular dan dijalankan secara terpisah tanpa terintegrasikan dengan modul yang lainnya. Pengujian integritas untuk mengetahui apakah modul dapat diuji ketika diintegrasikan dengan modul lainnya, dan pengujian transaksi data digunakan untuk mengatur jalannya transaksi data.

5.2.1. Pengujian Fungsionalitas

Pengujian fungsionalitas sistem dilakukan dengan menyiapkan sejumlah skenario sebagai tolak ukur keberhasilan pengujian. Pengujian fungsionalitas dilakukan dengan mengacu pada kasus penggunaan yang telah dijelaskan pada subbab 3.1.5. pengujian ini bertujuan untuk menguji apakah modul yang dibuat dapat diterima pada berkas pengembangan Unity secara modular atau terpisah dari berkas pengembangan *game* Food Merchant Saga.

5.2.1.1. Pengujian Meminta Data Festival yang Aktif

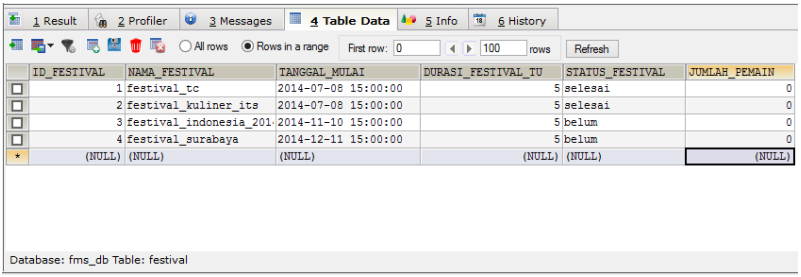
Berikut ini merupakan pembahasan proses pengujian untuk meminta data festival yang aktif. Sekenario pengujian dari pengujian meminta data festival yang aktif sebagaimana Tabel 5.1. Dokumentasi pengujian sebagaimana Gambar 5.1, Gambar 5.2 dan Gambar 5.3.

Tabel 5.1 Skenario Pengujian Meminta Data Festival yang Aktif

Nama Pengujian	Pengujian Meminta Data Festival yang Aktif
Kode	PF-001
Kode kasus penggunaan	UC-P03
Tujuan	Mengambil data festival yang aktif di server kemudian ditampilkan pada festival *list* di *client*
Kondisi awal	Pemain masuk ke halaman daftar festival
Skenario	1. Pemain masuk ke halaman daftar festival tanpa tersambung oleh *server* 2. Pemain masuk ke halaman daftar festival dengan tersambung oleh *server*

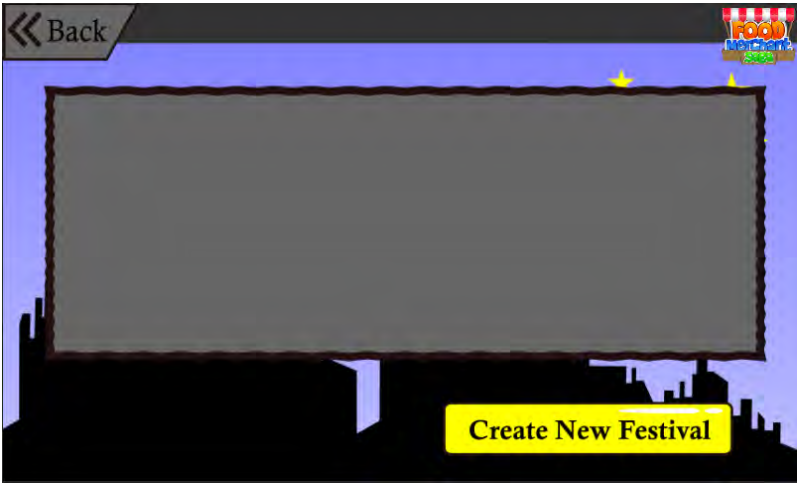
Masukan	-
Keluaran yang diharapkan	Daftar festival yang aktif pada basis data *server*
Hasil pengujian	Berhasil

Berikut merupakan isi tabel festival pada *server*. Pada tabel tersebut terdapat 3 kondisi sebagai status dari festival tersebut. Jika status festival "belum" berarti festival belum dimulai, serta akan ditampilkan pada halaman daftar festival. Selain itu, jika status festival adalah "mulai", maka festival tersebut sedang berjalan. Dan jika status festival "selesai" maka festival tersebut sudah berakhir.



Gambar 5.1 Isi Tabel Festival pada Basisdata *Server*

Jika *game* dijalankan tanpa ada koneksi dengan *server*. Sehingga akan ada tampilan seperti pada Gambar 5.1. Kemudian halaman dipanggil dengan ada koneksi internet, sehingga akan muncul data festival yang aktif seperti pada Gambar 5.2.



Gambar 5.2 Halaman Daftar Festival Tanpa Koneksi

Gambar 5.3 Halaman Daftar Festival dengan Koneksi

5.2.1.2. Pengujian Membuat Festival Baru

Berikut ini merupakan pembahasan proses pengujian untuk membuat festival baru. Sekenario pengujian dari pengujian membuat festival baru sebagaimana Tabel 5.2. Dokumentasi pengujian sebagaimana Gambar 5.4, Gambar 5.5, dan Gambar 5.6.

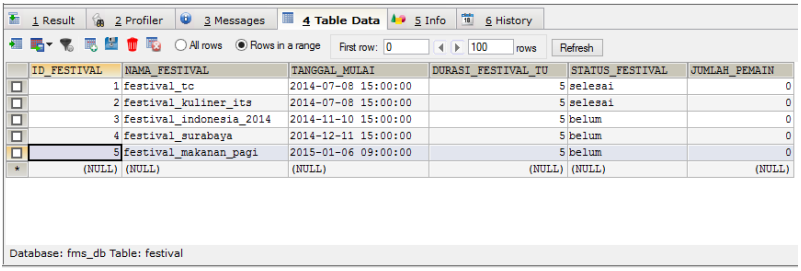
Tabel 5.2 Skenario Pengujian Membuat Festival Baru

Nama Pengujian	Pengujian Membuat Festival Baru
Kode	PF-002
Kode kasus penggunaan	UC-P01
Tujuan	Membuat atau mengadakan festival baru
Kondisi awal	Pemain masuk ke halaman membuat festival baru
Skenario	1. Pemain masuk ke halaman membuat festival baru 2. Pemain mengisi data yang dibutuhkan
Masukan	Nama festival, tanggal festival, waktu mulai
Keluaran yang diharapkan	Data festival berhasil disimpan di basis data *server* serta dapat terlihat pada halaman daftar festival aktif
Hasil pengujian	Berhasil

Berdasarkan Gambar 5.1, festival yang masih aktif berjumlah 2. Pemain akan menambahkan 1 festival lagi yang bisa untuk diikuti. Kemudian akan terjadi perubahan data pada tabel festival yang dapat ditunjukkan pada Gambar 5.5 dan Gambar 5.6.

Gambar 5.4 Membuat Festival Baru

Gambar 5.5 Membuat Festival Baru



Gambar 5.6 Isi Tabel Festival yang Baru

5.2.1.3. Pengujian Mengikuti Festival

Berikut ini merupakan pembahasan proses pengujian untuk mengikuti festival. Sekenario pengujian dari pengujian mengikuti festival sebagaimana Tabel 5.3. Dokumentasi pengujian sebagaimana Gambar 5.7 dan Gambar 5.8.

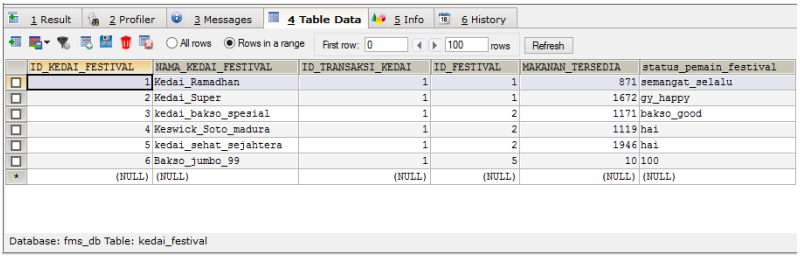
Tabel 5.3 Skenario Pengujian Mengikuti Festival

Nama Pengujian	Pengujian Mengikuti Festival
Kode	PF-003
Kode kasus penggunaan	UC-P02
Tujuan	Pemain mengikuti festival yang aktif
Kondisi awal	Pemain masuk ke halaman membuat festival baru
Skenario	1. Pemain masuk ke halaman mengikuti festival 2. Pemain mengisi data yang dibutuhkan
Masukan	Nama kedai pemain, kedai
Keluaran yang diharapkan	Data kedai pemain berhasil disimpan di basis data *server*
Hasil pengujian	Berhasil

Pada saat pemain memilih festival yang aktif, maka pemain perlu memasukkan data kedai yang diikutkan. Pemain perlu mengisi data seperti ditunjukkan pada Gambar 5.7.

Gambar 5.7 Pengisian Data Kedai Festival

Setelah itu, sistem langsung mengirim data ke *server* untuk disimpan pada basisdata seperti ditunjukkan pada Gambar 5.8.



1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info | 6 History

All rows | Rows in a range | First row: 0 | 100 rows | Refresh

ID_KEDAI_FESTIVAL	NAMA_KEDAI_FESTIVAL	ID_TRANSAKSI_KEDAI	ID_FESTIVAL	MAKANAN_TERSEDIA	status_pemain_festival
1	Kedai_Ramadhan	1	1	871	semangat_selalu
2	Kedai_Super	1	1	1672	gy_happy
3	kedai_bakso_spesial	1	2	1171	bakso_good
4	Keswick_Soto_madura	1	2	1119	hai
5	kedai_sehat_sejahtera	1	2	1946	hai
6	Bakso_jumbo_99	1	5	10	100
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Database: fms_db Table: kedai_festival

Gambar 5.8 Pengisian Data Kedai Festival

5.2.1.4. Pengujian Memperbarui Data ke *Server* pada Saat Festival Berlangsung

Berikut ini merupakan pembahasan proses pengujian untuk memperbarui data ke *server* pada saat festival berlangsung. Sekenario pengujian dari pengujian memperbarui data ke *server* pada saat festival berlangsung sebagaimana Tabel 5.4. Dokumentasi pengujian sebagaimana Gambar 5.9, Gambar 5.10, Gambar 5.11, Gambar 5.12, dan Gambar 5.13.

Tabel 5.4 Skenario Pengujian Memperbarui Data ke *Server* pada Saat Festival Berlangsung

Nama Pengujian	Pengujian Memperbarui Data ke *Server* pada Saat Festival Berlangsung
Kode	PF-004
Kode kasus penggunaan	UC-P04
Tujuan	Data pada basisdata *server* dapat diperbarui secara berkala
Kondisi awal	Pemain masuk ke halaman *gameplay* festival
Skenario	1. Pemain mengetikkan pesan pada *message box* segingga isi pesan pemain dapat berubah
Masukan	Isi pesan pemain
Keluaran yang diharapkan	Data pesan pemain dapat diperbarui
Hasil pengujian	Berhasil

Agar festival dapat mulai, maka status festival pada tabel festival harus diubah menjadi “mulai”, seperti pada Gambar 5.9.

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info | 6 History

All rows | Rows in a range | First row: 0 | 100 rows | Refresh

	ID_FESTIVAL	NAMA_FESTIVAL	TANGGAL_MULAI	DURASI_FESTIVAL_TU	STATUS_FESTIVAL	JUMLAH_PEMAIN
	1	festival_tc	2014-07-08 15:00:00	5	selesai	0
	2	festival_kuliner_its	2014-07-08 15:00:00	5	selesai	0
	3	festival_indonesia_2014	2014-11-10 15:00:00	5	belum	0
	4	festival_surabaya	2014-12-11 15:00:00	5	belum	0
	5	festival_makanan_pagi	2015-01-06 09:00:00	5	mulai	0
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Database: fms_db Table: festival

Gambar 5.9 Isi Tabel Festival

Kemudian pemain masuk pada *gameplay* Festival seperti pada Gambar 5.10. Lalu dapat dilihat juga perubahan data pemain pada festival akan selalu diperbarui di basisdata *server*.

Gambar 5.10 Uji Coba pada *Gameplay* Keadaan 1

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info | 6 History

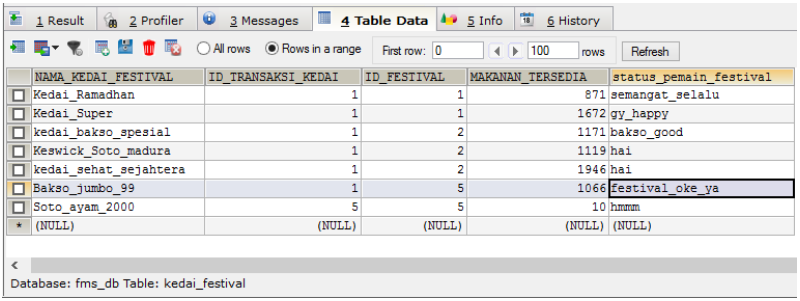
All rows | Rows in a range | First row: 0 | 100 rows | Refresh

	NAMA_KEDAI_FESTIVAL	ID_TRANSAKSI_KEDAI	ID_FESTIVAL	MAKANAN_TERSEDIA	status_pemain_festival
	Kedai_Ramadhan	1	1	871	semangat_selalu
	Kedai_Super	1	1	1672	gy_happy
	kedai_bakso_spesial	1	2	1171	bakso_good
	Keswick_Soto_madura	1	2	1119	hai
	kedai_sehat_sejahtera	1	2	1946	hai
	Bakso_jumbo_99	1	5	1051	pagi_yang_cerah
	Soto_ayam_2000	5	5	10	hmmm
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Database: fms_db Table: kedai_festival

Gambar 5.11 Isi kolom Status Pemain Keadaan 1

Gambar 5.12 Uji Coba *Gameplay* Keadaan 2



Gambar 5.13 Isi Kolom Status Pemain Keadaan 2

5.2.1.5. Pengujian Mengolah Data dari *Server* pada Saat Festival Berlangsung

Berikut ini merupakan pembahasan proses pengujian untuk mengolah data dari *server* pada saat festival berlangsung. Sekenario pengujian dari pengujian mengolah data dari *server* pada saat festival berlangsung sebagaimana Tabel 5.5. Dokumentasi pengujian sebagaimana Gambar 5.14, Gambar 5.15, Gambar 5.16, dan Gambar 5.17.

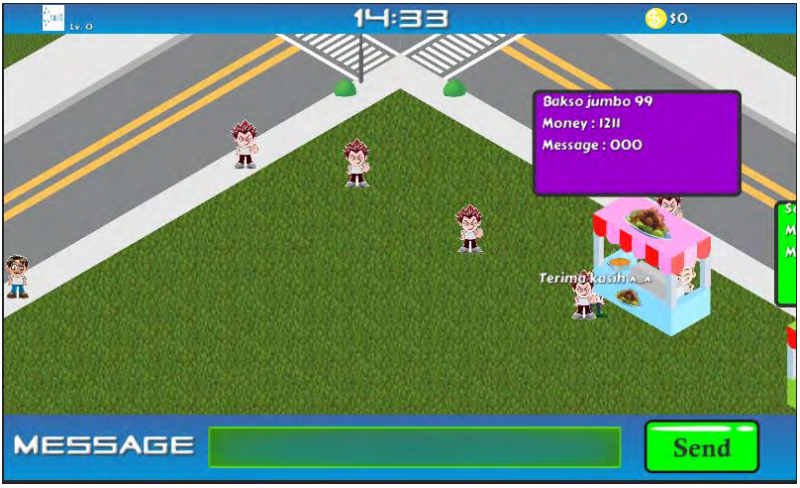
Tabel 5.5 Skenario Pengujian Mengolah Data dari *Server* pada Saat Festival Berlangsung

Nama Pengujian	Pengujian Mengolah Data dari *Server* pada Saat Festival Berlangsung
Kode	PF-005
Kode kasus penggunaan	UC-P05
Tujuan	Data pada basisdata *server* dapat diambil kemudian dijadikan data informasi pemain lain yang tergabung dalam satu festival
Kondisi awal	Pemain masuk ke halaman *gameplay* festival
Skenario	1. Pemain 1 melakukan perubahan data 2. Pemain 2 dapat melihat perubahan data dari pemain 1

Masukan	Pemain 1 mengganti isi pesan
Keluaran yang diharapkan	Data uang dan pesan pemain 1 dapat diketahui pemain 2
Hasil pengujian	Berhasil

Pemain 1 ditunjukkan dengan warna hijau, sedangkan pemain 2 ditunjukkan dengan warna ungu.

Gambar 5.14 Tampilan *Gameplay* pada Pemain 1



Gambar 5.15 Tampilan *gameplay* Pada Pemain 2

Gambar 5.16 Pemain 1 Melakukan Perubahan Data

Gambar 5.17 Pemain 2 Mengetahui Perubahan Data Pemain 1

5.2.2. Pengujian Integritas

Pengujian integritas sistem dilakukan dengan menyiapkan sejumlah skenario sebagai tolak ukur keberhasilan pengujian. Pengujian integritas dilakukan dengan mengacu pada kasus penggunaan yang telah dijelaskan pada subbab 3.1.5. Pengujian ini bertujuan untuk menguji apakah modul yang dibuat dapat terintegrasi dengan modul lainnya pada pengembangan *game* Food Merchant Saga.

5.2.2.1. Pengujian Integrasi dengan modul AI

Berikut ini merupakan pembahasan proses pengujian integrasi dengan modul AI. Sekenario pengujian dari pengujian integrasi dengan modul AI sebagaimana Tabel 5.6. Dokumentasi pengujian sebagaimana Gambar 5.18 dan Gambar 5.19.

Tabel 5.6 Skenario Pengujian Integrasi dengan Modul AI

Nama Pengujian	Pengujian Integrasi dengan Modul AI
Kode	PF-006
Kode kasus penggunaan	-
Tujuan	Melihat perilaku NPC yang menggunaan algoritma AI pada saat festival berlangsung
Kondisi awal	-
Skenario	1. Festival berlangsung tanpa ada kedai pemain 2. Festival berlangsung dengan 2 kedai pemain
Masukan	-
Keluaran yang diharapkan	Perbedaan perilaku NPC dengan 2 macam kondisi festival
Hasil pengujian	Berhasil

Gambar 5.18 Perilaku NPC Saat Tidak ada Kedai Festival

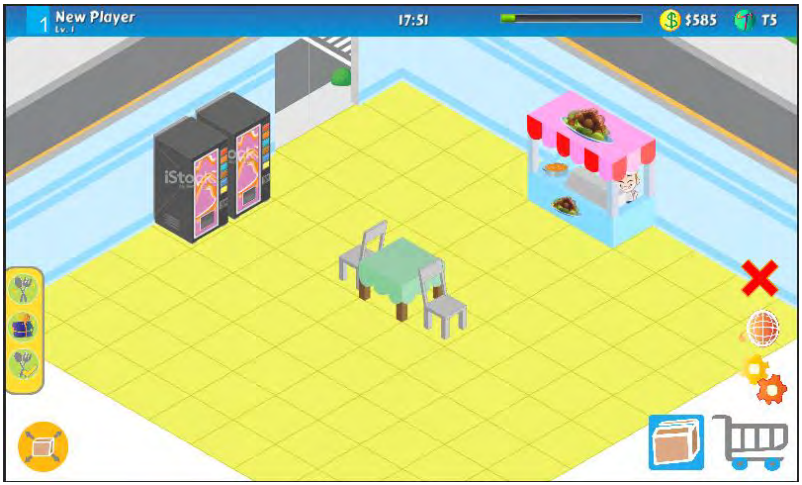
Gambar 5.19 Perilaku NPC pada Saat Ada Kedai Festival

5.2.2.2. Pengujian Integrasi dengan asset pemain pada permainan utama

Berikut ini merupakan pembahasan proses pengujian integrasi dengan asset pemain pada permainan utama. Skenario pengujian dari pengujian integrasi dengan asset pemain pada permainan utama sebagaimana Tabel 5.7. Dokumentasi pengujian sebagaimana Gambar 5.20 dan Gambar 5.17.

Tabel 5.7 Skenario Pengujian Integrasi dengan Asset Pemain pada Permainan Utama

Nama Pengujian	**Pengujian Integrasi dengan Asset Pemain pada Permainan Utama**
Kode	PF-007
Kode kasus penggunaan	-
Tujuan	Melihat dampak dan hubungan perubahan data permainan utama dengan modul *mini game*
Kondisi awal	Pemain hanya memiliki 1 kedai festival
Skenario	1. Pemain hanya memiliki 1 kedai festival, dengan ditunjukkan pada pilihan kedai pada saat ingin mengikuti festival 2. Pemain membeli kedai 2 pada permainan utama 3. Pemain membuat festival 2 dengan menggunakan kedai 2
Masukan	Data pemain dan kedai pemain
Keluaran yang diharapkan	Perbedaan data kedai yang dapat diikutkan pemain antara festival 1 dengan festival 2
Hasil pengujian	Berhasil



Gambar 5.20 *Gameplay* Permainan Utama dengan 1 Kedai

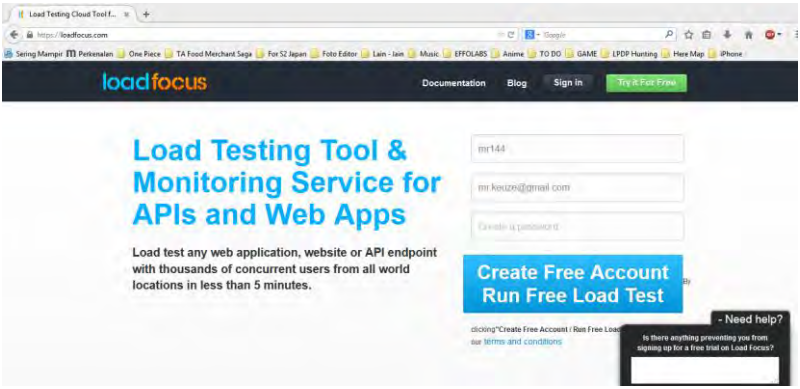
Gambar 5.21 Mengikuti Festival dengan Kedai 1

Gambar 5.22 *Gameplay* Permainan Utama dengan 2 Kedai

Gambar 5.23 Mengikuti Festival dengan Kedai 2

5.2.3. Pengujian Performa

Pengujian peforma dilakukan untuk mengetahui kemampuan sistem untuk menangani kebutuhan sumber daya berupa data dalam jumlah yang bervariasi. Pengujian ini dilakukan pada saat proses festival berlangsung. Dengan adanya pengujian ini dapat dilihat perbedaan waktu sinkronisasi data antar pemain. Pengujian ini akan dilakukan dengan menggunakan layanan yang disediakan pada situs loadfocus yang dapat diakses pada alamat http://loadfocus.com. halaman awal situs ini dapat dilihat pada Gambar 5.24.



Gambar 5.24 Halaman Depan Loadfocus

Pada proses sinkronisasi data dalam suatu festival, terdapat 2 fungsi yang dipanggil secara terus-menerus. Pertama adalah fungsi `UpdateStatusFestival()`, fungsi ini digunakan untuk memperbarui data pemain kedalam basisdata *server*. Kemudian yang kedua adalah fungsi `LoadDataStatus()`, fungsi ini digunakan untuk mengambil data seluruh pemian yang tergabung dalam festival. Oleh karena itu, pengujian performa yang dilakukan memanggil 2 fungsi tersebut dengan jumlah *user* atau pemain yang berbeda. Spesifikasi pengujian ini dijelaskan pada Tabel 5.8.

Tabel 5.8 Spesifikasi Pengujian Performa

Deskripsi Pengujian	Spesifikasi Pengujian Performa
Kode	PP-001
Alamat URI 1	http://effolabs.com/fms/index.php/FM/updatefest/stall_1/8/1190/this_is_status
Alamat URI 2	http://effolabs.com/fms/index.php/FM/loadstall/8
Jumlah Pengguna	2-10
Lokasi Pengguna	US East (Virginia Utara)
Durasi	60 detik
Waktu tunda	0.5 detik

Berdasarkan Tabel 5.8, dilakukanlah uji coba dengan menggunakan 2 sampai 10 pemain. Berikut adalah hasil uji coba performa dengan menggunakan loadfocus dengan pengujian pada URI 1 dan URI 2 sehingga didapatkan total waktu respon seperti pada Tabel 5.9.

Tabel 5.9 Hasil Uji Coba Performa

No	Jumlah Pemain	Total Waktu (ms)	Total *Hits/sec*	Ukuran(KB/*sec*)	Total Respon
1	2 pemain	587	7,53	3,77	450
2	3 pemain	574	11,29	5,66	621
3	4 pemain	594	14,42	7,25	808
4	5 pemain	599	18,03	9,05	1010
5	6 pemain	593	21,74	10,94	1215
6	7 pemain	625	23,0	11,86	229
7	8 pemain	616	27,27	13,75	1548
8	9 pemain	698	27,93	14,22	1629
9	10 pemain	720	31,07	15,61	1862

Berdasarkan Tabel 5.9, kolom total waktu merupakan rata-rata waktu yang dibutuhkan untuk respon transaksi data.Total *Hits/sec* merupakan rata-rata total transaksi data yang terjadi setiap detiknya. Ukuran merupakan ukuran data yang dikirim setiap detiknya. Dan total respon merupakan jumlah respon yang didapat aplikasi. Kesemua hasil merupakan akumulasi dari pengaksesan URI 1 dan URI 2 berdasarkan jumlah pemain yang berbeda.

5.3. Evaluasi Pengujian

Berdasarkan hasil pengujian fungsionalitas, pengujian integrasi, dan pengujian performa dilakukan evaluasi terhadap hasil pengujian tersebut sebagai berikut.

5.3.1. Evaluasi Pengujian Fungsionalitas

Pengujian fungsionalitas yang telah dilakukan pada subbab 5.2.1 memberikan hasil yang sesuai dengan skenario yang telah dibuat. Evaluasi pengujian masing-masing fungsionalitas dijelaskan sebagai berikut.

1. Fungsionalitas meminta data festival yang aktif berjalan sesuai dengan yang diharapkan
2. Fungsionalitas membuat festival baru berjalan sesuai dengan yang diharapkan
3. Fungsionalitas mengikuti festival berjalan sesuai dengan yang diharapkan
4. Fungsionalitas memperbarui data ke *server* pada saat festival berlangsung berjalan sesuai dengan yang diharapkan
5. Fungsionalitas mengolah data dari *server* pada saat festival berlangsung berjalan sesuai dengan yang diharapkan

Dari evaluasi di atas didapatkan bahwa semua fungsionalitas yang diuji pada sistem ini, semua skenario pengujian fungsional telah berhasil dilakukan serta fungsionalitas sistem telah berjalan sesuai yang diharapkan.

5.3.2. Evaluasi Pengujian Integritas

Pengujian integrasi yang telah dilakukan pada subbab 5.2.2 memberikan hasil yang sesuai dengan skenario yang telah dibuat. Evaluasi pengujian integrasi pada masing-masing modul dijelaskan sebagai berikut.

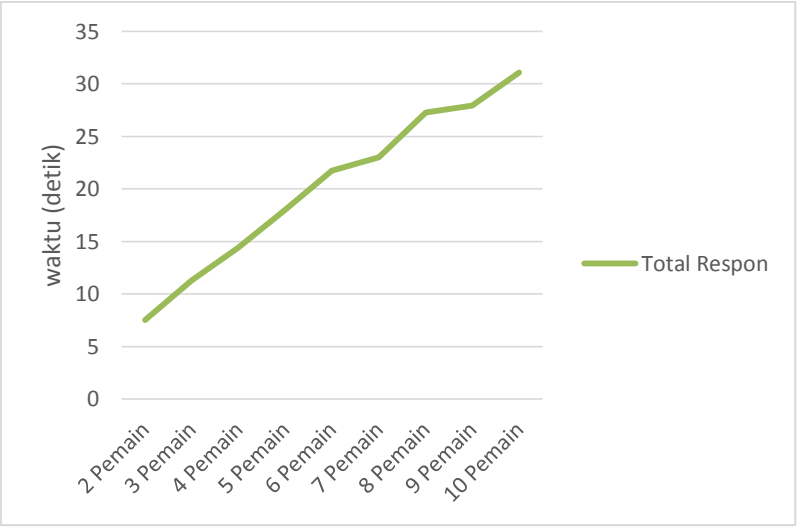
1. Integrasi sistem dengan modul AI berjalan sesuai dengan yang diharapkan
2. Integrasi sistem dengan asset pemain pada permainan utama berjalan sesuai dengan yang diharapkan

Dari evaluasi di atas didapatkan bahwa semua skenario pengujian integrasi untuk mengetahui kebenaran dan integrasi modul yang dikerjakan berhasil dilakukan. Sistem telah berjalan sesuai dengan yang diharapkan.

5.3.3. Evaluasi Pengujian Performa

Pengujian performa yang telah dilakukan pada subbab 5.2.3 memberikan hasil yang berbeda sesuai dengan jumlah pemain yang tergabung dalam festival. Kemampuan respon pada festival dapat dilihat dalam kolom *Hits/sec*, dimana kolom tersebut menunjukkan rata-rata transaksi data yang terjadi setiap detiknya. Hasil tersebutlah yang memberikan gambaran terhadap waktu sinkronisasi yang dibutuhkan ketika festival berlangsung. Diagram waktu respon dapat dilihat pada Gambar 5.25.

Dari Gambar 5.25 didapatkan bahwa semua skenario pengujian performa dengan jumlah pemain yang berbeda mempunyai hasil yang berbeda pula. Hal ini menunjukkan terdapat hubungan jumlah pemain dengan waktu sinkronisasi yang dibutuhkan. Semakin banyak jumlah pemain maka waktu sinkronisasi yang dibutuhkan semakin lama.



Gambar 5.25 Diagram Waktu Respon Sinkronisasi

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

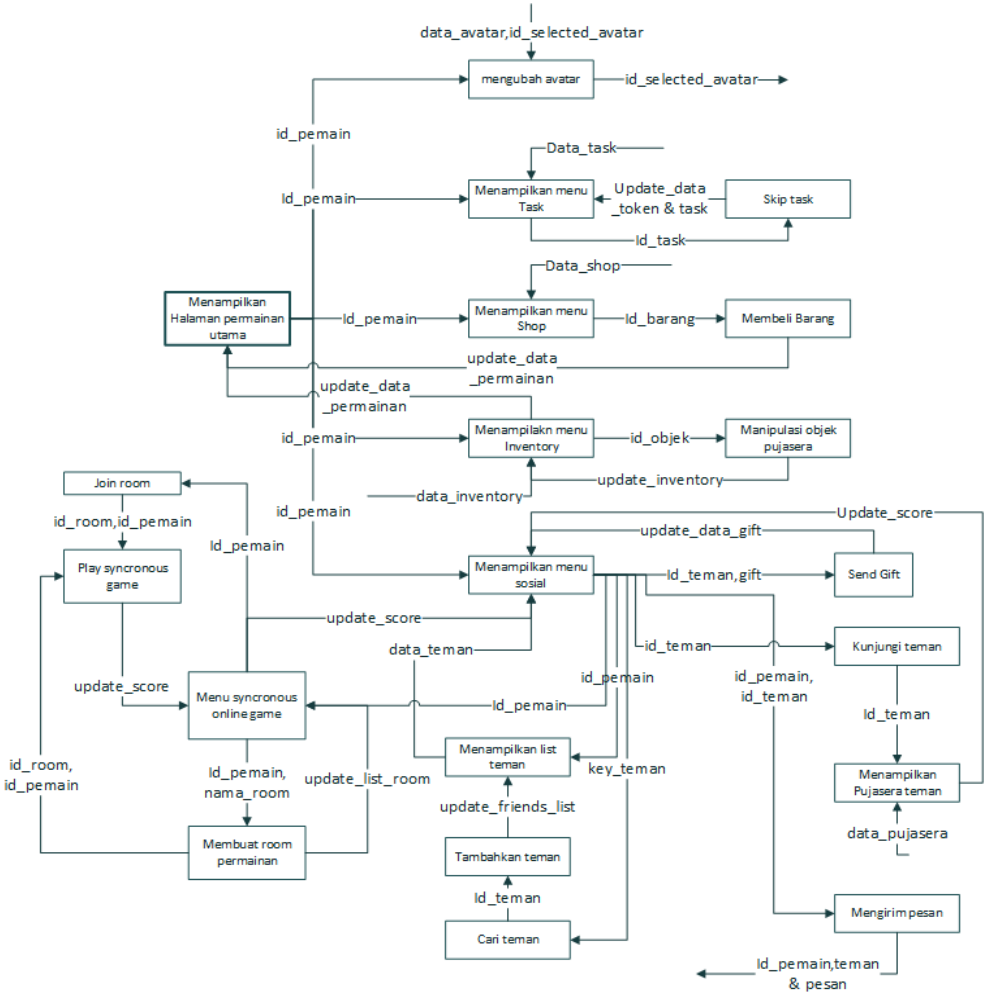
1. Proses sinkronisasi antar pemain pada *Mini Synchronous Game* Food Fest dalam Game Sosial Food Merchant Saga dapat berjalan dengan baik
2. Implementasi *gameplay* dan aturan permainan dapat berjalan sesuai dengan skenario yang dirancang.
3. Integrasi dengan modul permainan utama, modul *web service*, dan modul kecerdasan buatan dapat berjalan dengan baik.
4. Implementasi *Mini Synchronous Game* Food Fest dapat berjalan dengan baik pada perangkat android.
5. Perbedaan jumlah pemain pada festival dapat mempengaruhi waktu sinkronisasi data.

6.2. Saran

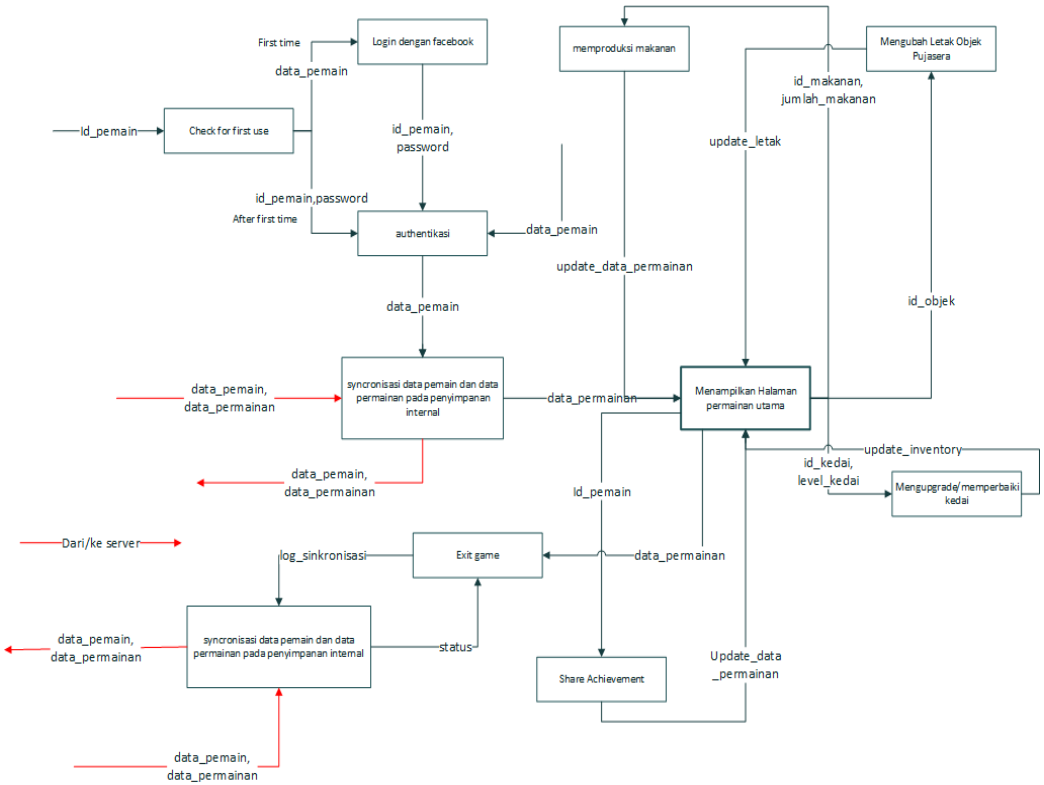
Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Diantaranya adalah sebagai berikut:

1. *Gameplay* dibuat lebih variatif dan perlu ditambah fitur-fitur yang lain.
2. Desain lebih disempurnakan agar menambah daya tarik terhadap *game*.

LAMPIRAN A
DIAGRAM BLOK FOOD MERCHANT SAGA



Gambar A.1 Diagram Blok Bagian Pertama Aplikasi Permainan Food Merchant Saga Bagian Pertama



Gambar A.2 Diagram Blok Bagian Pertama Aplikasi Permainan Food Merchant Saga Bagian Kedua

LAMPIRAN B
KODE SUMBER

```
using UnityEngine;
using System.Collections;
public class FFFestival : MonoBehaviour {
	public int idFestival;
	public string namaFestival;
	public string tanggalMulai;
	public int durasiFestival;
	public string statusFestival;
	public int jumlahPemain;
	int IdFestival {
		get {
			return this.idFestival;
		}
		set {
			idFestival = value;
		}
	}
	string NamaFestival {
		get {
			return this.namaFestival;
		}
		set {
			namaFestival = value;
		}
	}
	string TanggalMulai {
		get {
			return this.tanggalMulai;
		}
		set {
			tanggalMulai = value;
		}
	}
	int DurasiFestival {
		get {
			return this.durasiFestival;
		}
		set {
```

```
                        durasiFestival = value;
                }
        }
        string StatusFestival {
                get {
                        return this.statusFestival;
                }
                set {
                        statusFestival = value;
                }
        }
        int JumlahPemain {
                get {
                        return this.jumlahPemain;
                }
                set {
                        jumlahPemain = value;
                }
        }
}
```

Kode Sumber B.1 Kelas `FFFestival`

```
using UnityEngine;
using System.Collections;
using Assets.Scripts.FMSClass;
using MiniJSON;
using Assets.Scripts.Utils;
using System.Collections.Generic;
using Assets.Scripts.FMSData;
using Assets.Scripts.FMSClass.Attribute;
using System;
using Assets.Scripts.Global;

public class FFMenuController : MonoBehaviour {

	public int status; //digunakan untuk menandai
status halaman yang muncul.
	//0. Passive, 1. Home menu, 2. Join Game, 3.
CreateNew, 4. joinFestivalPage

	//variabel umum
	FFWebService service = new FFWebService();

	//variabel for MainPage
	private bool statusIkutFestival;
	private int statusMulaiFestival;
	public FFKedaiFestival kedaiFestival = new
FFKedaiFestival();
	InternalStorageUtil festivalSave;
	FFMyFestivalData festivalData;

	//variabel for FestivalListPage
	public FFFestival festival = new FFFestival();
	ArrayList listFestival;

	GameObject btnPlay, btnPlayFestivalNow,
btnBack, btnLoad, btnCreate, btnStart, btnJoin;
	GameObject mainPage, festivalListPage,
createFestivalPage, passivePage, joinFestivalPage,
gameplay;

	public List<Stall> listStall;
```

```
        //lain lain
        string response, stat;
        public Vector2 scrollPosition;
        int rx, ry, rw, rh;
        UILabel tbNama, tbD, tbM, tbY, tbhh, tbmm,
tbTimes, tbStandName;
        FFDateTime dateTime = new FFDateTime();
        bool canSwitch = false;
        bool waitActive = false;
        bool backActive = true;
        int idFest;
        int jumlahStand, standStatus;

        public PlayerData PlayerData
        {
                get { return Managers.Data.PlayerData; }
                set { Managers.Data.PlayerData = value; }
        }

        // Use this for initialization
        public void Start () {
                Init ();
                initKedaiPemain ();
                LoadDataFestival ();
                UIEventListener.Get (GameObject.Find
("Button - Back")).onClick += this.OnClickBack;
                UIEventListener.Get (GameObject.Find
("Button - Play")).onClick += this.OnClickPlay;
                UIEventListener.Get (GameObject.Find
("Button - Play Festival Now")).onClick +=
this.OnClickPlayFestivalNow;

        UIEventListener.Get(GameObject.Find("Button -
Create")).onClick += this.OnClickCreate;

        UIEventListener.Get(GameObject.Find("Button -
Start")).onClick += this.OnClickStart;
                UIEventListener.Get (GameObject.Find
("Button - Join Now")).onClick += this.OnClickJoin;
```

```
                UIEventListener.Get (GameObject.Find
("Button - Stand Left")).onClick += this.stleft;
                UIEventListener.Get (GameObject.Find
("Button - Stand Right")).onClick += this.stright;
        }

        void Init ()
        {
                status = 1;
                //button
                btnLoad = GameObject.Find ("Button -
Loading");
                btnPlay = GameObject.Find ("Button -
Play");
                btnPlayFestivalNow = GameObject.Find
("Button - Play Festival Now");
                btnCreate = GameObject.Find ("Button -
Create");
                btnStart = GameObject.Find ("Button -
Start");
                btnJoin = GameObject.Find ("Button - Join
Now");
                btnPlayFestivalNow.transform.Translate(0,
20,0);

                //page
                mainPage = GameObject.Find ("MainPage");
                festivalListPage = GameObject.Find
("FestivalListPage");
                createFestivalPage = GameObject.Find
("CreateFestivalPage");
                passivePage = GameObject.Find
("PassivePage");
                joinFestivalPage =
GameObject.Find("JoinFestivalPage");

                rx = Screen.width * 55 / 800;
                ry = Screen.height * 90 / 480;
                rw = Screen.width * 17 / 20;
                rh = Screen.height * 13 / 24;
```

```
            tbNama = GameObject.Find("UILabel -
name").GetComponent<UILabel>();
            tbD = GameObject.Find("UILabel -
day").GetComponent<UILabel>();
            tbM = GameObject.Find("UILabel -
month").GetComponent<UILabel>();
            tbY = GameObject.Find("UILabel -
year").GetComponent<UILabel>();
            tbTimes = GameObject.Find("UILabel -
times").GetComponent<UILabel>();
            tbStandName = GameObject.Find("UILabel -
stand name").GetComponent<UILabel>();
            SetDateTime ();
      }

      void initKedaiPemain ()
      {
            jumlahStand = 0;
            foreach (FMSObject d in
Managers.Data.ListUsedFMS)
            {
                  if(d is Stall)
                  {
                        Stall a = d as Stall;
                        GameObject o
=(GameObject)Instantiate(Loader.Load(a.PrefabName));

      o.transform.Translate(2+(15*jumlahStand), 10,
0);
                        listStall.Add(a);
                        jumlahStand++;
                        standStatus=1;
                  }
            }
      }

      void stleft (GameObject go)
      {
            if(standStatus>1)
```

```
		{
			Debug.Log ("stand
status"+standStatus);
			foreach(Stall o in listStall)
			{

	GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (15, 0, 0);
			}
			standStatus--;
		}
	}

	void stright (GameObject go)
	{
		Debug.Log ("stand status"+standStatus);
		if(standStatus<jumlahStand)
		{
			foreach(Stall o in listStall)
			{

	GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (-15, 0, 0);
			}
			standStatus++;
		}
	}

	void OnClickBack (GameObject go)
	{
		if (status == 1 && backActive) {
			MoveFromMainPage();
		}else if(status == 2 && backActive)
		{
			MoveFromFestivalListPage();
		}else if(status == 3 && backActive)
		{
			MoveFromCreateFestivalPage();
		}else if(status == 4 && backActive)
		{
```

```
                        MoveFromJoinFestivalPage();
                }
        }

        void OnClickPlay (GameObject go)
        {
                status = 2;
                MoveFromMainPage ();
        }

        void OnClickPlayFestivalNow (GameObject go)
        {
                Application.LoadLevel
("GameplayFestival");
        }

        void OnClickCreate (GameObject go)
        {
                if (statusIkutFestival == false) {
                        status = 3;
                        MoveFromFestivalListPage ();
                }
        }

        int a = 0;
        void OnClickStart (GameObject go)
        {
                status = 4;
                btnStart.transform.Translate (0, 20, 0);
                btnJoin.transform.Translate (20, 0, 0);
                btnLoad.transform.Translate (0, -20, 0);
                if (!waitActive) {
                        StartCoroutine (Wait ());
                }
        }

        void OnClickJoin (GameObject go)
        {
                status = 1;
                btnStart.transform.Translate (0, 20, 0);
```

```
                btnJoin.transform.Translate (20, 0, 0);
                btnLoad.transform.Translate (0, -20, 0);
                if (!waitActive) {
                        StartCoroutine (Wait ());
                }
        }

        void MoveFromMainPage ()
        {
                if (status == 1) {

        Application.LoadLevel("GamePlayTest");
                } else if (status == 2) {
                        mainPage.transform.Translate (0,
20, 0);

        btnPlayFestivalNow.transform.Translate (0, -20,
0);

        festivalListPage.transform.Translate (-20, 0,
0);

        StartCoroutine("ILoadFestivalList");
                }
        }

        void MoveFromFestivalListPage ()
        {
                if (status == 2) {
                        mainPage.transform.Translate (0, -
20, 0);

        btnPlayFestivalNow.transform.Translate (0, 20,
0);

        festivalListPage.transform.Translate (20, 0,
0);
                        status = 1;
                        LoadDataFestival ();
                } else if (status == 3) {
```

```
        festivalListPage.transform.Translate (20, 0,
0);

        createFestivalPage.transform.Translate (20, 0,
0);
                } else if (status == 4) {

        festivalListPage.transform.Translate (20, 0,
0);

        joinFestivalPage.transform.Translate (0, 20,
0);
                        foreach(Stall o in listStall)
                        {

        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, -13, 0);
                        }
                }
        }

        void MoveFromCreateFestivalPage ()
        {
                if (status == 3) {

        festivalListPage.transform.Translate (-20, 0,
0);

        createFestivalPage.transform.Translate (-20, 0,
0);
                        status=2;
                } else if (status == 4) {

        createFestivalPage.transform.Translate (-20, 0,
0);

        joinFestivalPage.transform.Translate (0, 20,
0);
                        foreach(Stall o in listStall)
```

```
                    {

        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, -13, 0);
                    }
                }
        }

        void MoveFromJoinFestivalPage ()
        {
                if (status == 4) {

        joinFestivalPage.transform.Translate (0, -20,
0);

        festivalListPage.transform.Translate (-20, 0,
0);
                    foreach(Stall o in listStall)
                    {

        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, 13, 0);
                    }
                    status=2;
                } else if (status == 1) {

        joinFestivalPage.transform.Translate (0, -20,
0);
                    mainPage.transform.Translate (0, -
20, 0);

        btnPlayFestivalNow.transform.Translate (0, 20,
0);
                    foreach(Stall o in listStall)
                    {

        GameObject.Find(o.PrefabName+"(Clone)").transfo
rm.Translate (0, 13, 0);
                    }
                    LoadDataFestival ();
```

```
		}
	}

	void SetDateTime ()
	{
		tbD.text = dateTime.GetDay ();
		tbM.text = dateTime.GetMonth ();
		tbY.text = dateTime.GetYear ();
		tbTimes.text = dateTime.GetTime ();
	}

	void LoadDataFestival ()
	{
		festivalSave = new InternalStorageUtil
("festivalSave.dat");
		festivalData = new FFMyFestivalData ();
		if(!festivalSave.IsEmpty)
		{
			festivalData =
(FFMyFestivalData)festivalSave.Load();
			idFest = festivalData.idFestival;
			if(festivalData.idFestival != 0)
				statusIkutFestival = true;
			else
			{
				statusIkutFestival = false;
			}
		}
		else
		{
			statusIkutFestival = false;
		}
	}

	void ResetDataFestival ()
	{
		festivalSave = new InternalStorageUtil
("festivalSave.dat");
		festivalData = new FFMyFestivalData ();
		festivalData.idFestival = 0;
```

```
                festivalData.namaKedai = "";
                festivalData.idKedai = 0;
                festivalSave.Save (festivalData);
                statusMulaiFestival = 0;
                statusIkutFestival = false;
        }

        void SaveDataFestival (int idF, string namaK,
int idK)
        {
                festivalSave = new InternalStorageUtil
("festivalSave.dat");
                festivalData = new FFMyFestivalData ();
                festivalData.idFestival = idF;
                festivalData.namaKedai = namaK;
                festivalData.idKedai = idK;
                festivalSave.Save (festivalData);
                statusIkutFestival = true;
        }

        void OnGUI(){
                if (status == 2) {
                        string namaFest;
                        GUILayout.BeginArea (new Rect (rx,
ry, rw, rh));
                        scrollPosition =
GUILayout.BeginScrollView(scrollPosition,
GUILayout.Width(rw), GUILayout.Height(rh));
                        if(listFestival!=null)
                                for(int i=0;
i<listFestival.Count; i++)
                                {
                                        FFFestival fests;
                                        fests =
listFestival[i] as FFFestival;
                                        namaFest =
fests.namaFestival.Replace("_", " ");

        GUILayout.Label("Festival Id : " +
fests.idFestival);
```

```
        GUILayout.Label("Festival Name : " + namaFest);

        GUILayout.Label("Date Time          : " +
fests.tanggalMulai);

        if(GUILayout.Button("Join Festival") &&
statusIkutFestival==false)
                                        {

        idFest=fests.idFestival;
                                                status=4;

        MoveFromFestivalListPage();
                                        }
                                }

                        GUILayout.EndScrollView();
                        GUILayout.EndArea ();
                }
        }

        int clock=0;
        int mulai;
        // Update is called once per frame
        void Update () {
                clock++;
                if (clock == 120)
                {
                        clock=0;

        print("statusIkutFestival="+statusIkutFestival)
;
                        if(statusIkutFestival == true)
                        {

        StartCoroutine("ICekStatusFestival");
                                if(statusMulaiFestival == 1
&& mulai==0 && status == 1)
                                {
```

```
        //btnPlayFestivalNow.transform.position = new
Vector3(127.5356, -112.4831, 1368.384);

        btnPlay.transform.Translate(0, 20, 0);

        btnPlayFestivalNow.transform.Translate(0, -20,
0);
                                        mulai=1;
                                }else
if(statusMulaiFestival == 0 && mulai==1 && status ==
1)
                                {

        btnPlay.transform.Translate(0, -20, 0);

        btnPlayFestivalNow.transform.Translate(0, 20,
0);
                                        mulai=0;
                                }
                        }
                        if(status==2)
                        {

        StartCoroutine("ILoadFestivalList");
                        }
                }
                if (canSwitch) {
                        canSwitch = false;
                        btnStart.transform.Translate (0, -
20, 0);
                        btnJoin.transform.Translate (-20,
0, 0);
                        btnLoad.transform.Translate (0,
20, 0);
                        if(status == 4)
                        {
                                StartCoroutine
("IAddFestival");
```

```
                                MoveFromCreateFestivalPage
();
                        }
                        else if(status==1)
                        {

        StartCoroutine("JoinFestival");
                                MoveFromJoinFestivalPage
();
                        }
                }
        }

        IEnumerator Wait(){
                waitActive = true;
                backActive = false;
                yield return new WaitForSeconds (3.0f);
                canSwitch = true;
                waitActive = false;
                backActive = true;
        }

        IEnumerator ICekStatusFestival()
        {
                WWW www = new
WWW(service.CEKSTATUSFESTIVAL+festivalData.idFestival)
;
                float elapsedTime = 0.0f;
                while (!www.isDone) {
                        elapsedTime += Time.deltaTime;
                        if (elapsedTime >= 10.0f) break;
                        yield return null;
                }
                if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
                        Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
                        yield break;
                }
                this.response = www.text;
```

```
                //print (response);
                IList search = (IList)
Json.Deserialize(response);

                foreach (IDictionary fest in search)
                {
                        stat =
(string)fest["STATUS_FESTIVAL"];
                }
                if (stat == "mulai") {
                print("mulai statuse");
                        statusMulaiFestival = 1;
                } else if (stat == "selesai") {
                print("selesai statuse");
                        ResetDataFestival();
                }
        }

        IEnumerator ILoadFestivalList ()
        {
                WWW www = new
WWW(service.LOADFESTIVALLIST);
                float elapsedTime = 0.0f;
                while (!www.isDone) {
                        elapsedTime += Time.deltaTime;
                        if (elapsedTime >= 10.0f) break;
                        yield return null;
                }
                if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
                        Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
                        yield break;
                }
                this.response = www.text;
                IList search = (IList)
Json.Deserialize(response);
                listFestival = new ArrayList ();
                foreach (IDictionary fest in search)
                {
```

```
                    festival = new FFFestival();
                    //menampung ke festival
                    festival.idFestival =
int.Parse((string)fest["id_festival"]);
                    festival.namaFestival =
(string)fest["nama_festival"];
                    festival.tanggalMulai =
(string)fest["tanggal_mulai"];
                    festival.jumlahPemain =
int.Parse((string)fest["jumlah_pemain"]);
                    listFestival.Add(festival);
                }
        }

        IEnumerator IAddFestival()
        {
                string namaFest, dateFest, times;
                string yy, mm, dd;
                namaFest = tbNama.text;
                yy = tbY.text;
                mm = tbM.text;
                dd = tbD.text;
                if(tbTimes.text=="09 AM")
                        times = "09_00_00";
                else
                        times = "15_00_00";
                namaFest = namaFest.Replace(" ", "_");
                dateFest = string.Concat
(yy,"_",mm,"_",dd);
                //print
(service.ADDFESTIVAL+namaFest+"/"+dateFest+"/"+times);
                WWW www = new
WWW(service.ADDFESTIVAL+namaFest+"/"+dateFest+"/"+time
s);
                float elapsedTime = 0.0f;
                while (!www.isDone) {
                        elapsedTime += Time.deltaTime;
                        if (elapsedTime >= 10.0f) break;
                        yield return null;
                }
```

```
            if (!www.isDone || !string.IsNullOrEmpty
(www.error)) {

      Debug.LogError (string.Format ("Fail
Whale!\n{0}", www.error));
                                          yield break;
                           }
            string response = www.text;
            IList search = (IList)
Json.Deserialize(response);
            listFestival = new ArrayList ();

            foreach (IDictionary fest in search)
            {
                  this.idFest =
int.Parse((string)fest["message"]);
            }
      }

      IEnumerator JoinFestival() {
            //Managers.Debug.print ("Masuk");
            string idPlayer = PlayerData.playerid;
            tbStandName.text =
tbStandName.text.Replace (" ", "_");
            WWW www = new
WWW(service.JOINFESTIVAL+tbStandName.text+"/"+idPlayer
+"/"+standStatus+"/"+idFest);
            //Managers.Debug.print
(service.JOINFESTIVAL + tbStandName.text + "/" +
idPlayer + "/" + standStatus + "/" + idFest);
            //print (service.JOINFESTIVAL +
tbStandName.text + "/" + idPlayer + "/" + standStatus
+ "/" + idFest);
            //Managers.Debug.print ("Masuk2");
            float elapsedTime = 0.0f;
            while (!www.isDone) {
                  elapsedTime += Time.deltaTime;
                  if (elapsedTime >= 10.0f) break;
                  yield return null;
            }
```

```
            if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
                    Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
                    yield break;
            }
            //Managers.Debug.print ("Masuk3");
            string response = www.text;
            //Managers.Debug.print (response);
            SaveDataFestival (idFest,
tbStandName.text, standStatus);
        }
}
```

Kode Sumber B.2 Kelas `FFMenuController`

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Assets.Scripts.FMSClass.Attribute;
using Facebook.MiniJSON;
using System;
using Assets.Scripts.Utils;
using Assets.Scripts.FMSClass;


public class FFGameplayManager : MonoBehaviour {

        private int idPemain;
        string myStatus = "000";

        InternalStorageUtil festivalSave;
        FFMyFestivalData festivalData;

        GameObject FFTile;
        public Transform tilePrefab;
        GameObject o;

        public double[,] arrayX;
        public double[,] arrayY;
        public GameObject npcObj;
        public GameObject btnStart;

        public List<object> listKedaiUrl = new
List<object> ();

        public string namaKedai;

        FFWebService service = new FFWebService();

        public List<FMSObject> listUsedFest = new
List<FMSObject> ();
        Food foodSatu = new Food("food_01", "Bakso
Kikil", 20, 5, 4, 1);
        Food foodDua = new Food("food_02", "Bakso Iga",
30, 9, 7, 2);
```

```
        Food foodTiga = new Food("food_03", "Bakso
Jumbo", 40, 14, 10, 3);

        FFKedaiPemain kedaiPemain;
        private ArrayList listKedai;
        public int curTile = 17;

        public string dataUrl;

        public GameObject dataCarrier;
        UILabel labelWaktu, labelStatus;

        string statusPemain;
        ArrayList listStatusPemain = new ArrayList();
        int sx, sy, sw, sh;

        UILabel msgStatus;

        GameObject p;

        private int jumlahPemain;

        // Use this for initialization
        void Start () {
                LoadDataFestival ();
                StartCoroutine ("LoadDataStatus");
                sx = Screen.width * 20 / 800;
                sy = Screen.height * 30 / 480;
                sw = Screen.width * 200 / 800;
                sh = Screen.height * 350 / 480;
                msgStatus = GameObject.Find("UILabel -
status fest").GetComponent<UILabel>();
                UIEventListener.Get (GameObject.Find
("Button - Send Status")).onClick += this.OnClickSend;
                labelWaktu = GameObject.Find("Label -
waktu festival").GetComponent<UILabel>();
                //labelStatus = GameObject.Find("Label -
Status").GetComponent<UILabel>();
                namaKedai = festivalData.namaKedai;
```

```
                FFTile =
GameObject.Find("FoodFestTiles");
                dataCarrier = GameObject.Find
("DataCarrier");
                TileInstantiate();
                loadKedaiFestivalServer ();
        }

        void LoadDataFestival ()
        {
                festivalSave = new InternalStorageUtil
("festivalSave.dat");
                festivalData = new FFMyFestivalData ();
                if(!festivalSave.IsEmpty)
                {
                        festivalData =
(FFMyFestivalData)festivalSave.Load();
                }
        }

        private void loadKedaiFestivalServer ()
        {
                WWW www = new
WWW("http://fms.effolabs.com/indeks.php/FM/loadstall/"
+festivalData.idFestival);

                while(!www.isDone)
                {
                        //Managers.Debug.print("belum
selesai...");
                }

                if (www.isDone &&
string.IsNullOrEmpty(www.error)) {
                        dataUrl = www.text;

                        string response = dataUrl;
                        listKedaiUrl = (List<object>)
Json.Deserialize(response);
                        listKedai = new ArrayList ();
```

```
                    int a = 1;
                    IDictionary ked;
                    for(int i=0; i<listKedaiUrl.Count;
i++)
                    {
                            ked =
(IDictionary)listKedaiUrl[i];

                            kedaiPemain = new
FFKedaiPemain();
                            kedaiPemain.namaKedai =
(string)ked["nama_kedai_festival"];
                            kedaiPemain.idKedai =
int.Parse((string)ked["id_kedai"]);

       kedaiPemain.makanan_tersedia =
int.Parse((string)ked["makanan_tersedia"]);
                            kedaiPemain.prefabKedai =
namaPrefabs(kedaiPemain.idKedai);
                            kedaiPemain.posisiKedaiX =
lokasiX(a);
                            kedaiPemain.posisiKedaiY =
lokasiY(a);
                            if(a==1 || a==2)

       kedaiPemain.statusPosisi=1;
                            else if(a==3 || a==4 ||
a==5)

       kedaiPemain.statusPosisi=2;
                            else if(a==6 || a==7 ||
a==8)

       kedaiPemain.statusPosisi=3;
                            else if(a==9 || a==10)

       kedaiPemain.statusPosisi=4;
                            listKedai.Add
(kedaiPemain);
                            a++;
```

```
                    }
                    loadKedaiFestival ();

            }//else{
                    //Managers.Debug.print("galat!");
            //}
    }

    int lokasiX(int ix)
    {
            if (ix == 1)
                    return 1;
            else if (ix == 2)
                    return 1;
            else if (ix == 3)
                    return 5;
            else if (ix == 4)
                    return 10;
            else if (ix == 5)
                    return 15;
            else if (ix == 6)
                    return 18;
            else if (ix == 7)
                    return 18;
            else if (ix == 8)
                    return 18;
            else if (ix == 9)
                    return 15;
            else
                    return 10;
    }

    int lokasiY(int iy)
    {
            if (iy == 1)
                    return 7;
            else if (iy == 2)
                    return 12;
            else if (iy == 3)
                    return 17;
```

```
		else if (iy == 4)
			return 17;
		else if (iy == 5)
			return 17;
		else if (iy == 6)
			return 12;
		else if (iy == 7)
			return 7;
		else if (iy == 8)
			return 2;
		else if (iy == 9)
			return 1;
		else
			return 1;
	}

	double LokasiStatusBoardX(int bx)
	{
		if (bx == 1)
			return 5.427101;
		else if (bx == 2)
			return 10.98856;
		else if (bx == 3)
			return 15.38226;
		else if (bx == 4)
			return 11.12339;
		else if (bx == 5)
			return 3.189144;
		else if (bx == 6)
			return -10.0962;
		else if (bx == 7)
			return -14.76414;
		else if (bx == 8)
			return -19.0656;
		else if (bx == 9)
			return -15.92847;
		else
			return -9.843557;
	}
```

```
	double LokasiStatusBoardY(int by)
	{
		if (by == 1)
			return 2.273319;
		else if (by == 2)
			return -0.2627442;
		else if (by == 3)
			return -5.413941;
		else if (by == 4)
			return -10.60577;
		else if (by == 5)
			return -15.01725;
		else if (by == 6)
			return -13.34508;
		else if (by == 7)
			return -9.806063;
		else if (by == 8)
			return -5.025074;
		else if (by == 9)
			return -1.656353;
		else
			return 1.279121;
	}

	string namaPrefabs(int id)
	{
		if(id==1)
			return "prefab_kedai_bakso";
		else
			return "prefab_kedai_soto";
	}

	void TileInstantiate ()
	{
		arrayX = new double[100,100];
		arrayY = new double[100,100];

		for(int i = 0; i<100; i++)
		{
			for(int j = 0; j<100; j++)
```

```
                        {
                                arrayX[i,j] = (0.89 * j) -
(0.89 * i);
                                arrayY[i,j] = 3 - (0.515 *
j) - (0.515 * i);
                        }
                }

                curTile = 20;
                for (int i = 0; i<curTile; i++)
                {
                        for (int j = 0; j<curTile; j++)
                        {
                                var tileInstant =
Instantiate(tilePrefab) as Transform;
                                tileInstant.position = new
Vector3((float)arrayX[i,j], (float)arrayY[i,j],
FFTile.transform.position.z);
                                tileInstant.parent =
FFTile.transform;
                                TileScript tilePos =
tileInstant.GetComponent<TileScript>();
                                tilePos.posX = i;
                                tilePos.posY = j;
                                tilePos.gridPosition = new
GridPosition(i, j);
                        }
                }
        }
        int gx=0, gy=0;
        void loadKedaiFestival ()
        {
                int i;
                for(i=0; i<listKedai.Count; i++)
                {
                        FFKedaiPemain ked;
                        ked = listKedai[i] as
FFKedaiPemain;
```

```
                    o
=(GameObject)Instantiate(Loader.Load(ked.prefabKedai))
; //game object kedai
                    if(ked.namaKedai ==
festivalData.namaKedai)
                        p =
(GameObject)Instantiate(Loader.Load("Label -
StatusPlayer"));
                    else
                        p =
(GameObject)Instantiate(Loader.Load("Label -
StatusOther"));
                    o.transform.position = new
Vector3((float)arrayX[ked.posisiKedaiX,ked.posisiKedai
Y], (float)arrayY[ked.posisiKedaiX,ked.posisiKedaiY],
FFTile.transform.position.z);
                    p.transform.position = new
Vector3((float)LokasiStatusBoardX(i+1),
(float)LokasiStatusBoardY(i+1), 0);

                    o.GetComponent<FMSView>().enabled
= false;

                    if(ked.statusPosisi==1){

      o.GetComponentInChildren<SpriteRenderer>().spri
te = o.GetComponentInChildren<SpriteScript>().sprite0;
                    }
                    else if(ked.statusPosisi==2){

      o.GetComponentInChildren<SpriteRenderer>().spri
te = o.GetComponentInChildren<SpriteScript>().sprite1;
                    }
                    else if (ked.statusPosisi==3){

      o.GetComponentInChildren<SpriteRenderer>().spri
te = o.GetComponentInChildren<SpriteScript>().sprite2;
                    }
                    else if (ked.statusPosisi==4){
```

```
        o.GetComponentInChildren<SpriteRenderer>().spri
te = o.GetComponentInChildren<SpriteScript>().sprite3;
                        }

                        listUsedFest.Add(new
Stall("kedai_01", ked.namaKedai, new
GridPosition(ked.posisiKedaiX,ked.posisiKedaiY), 90,
100, GridRotation.Rotate0, 1, 10, 1, 10,
ked.prefabKedai, new Food[] { foodSatu, foodDua,
foodTiga }, 10));

                        var idkStall =
listUsedFest[listUsedFest.Count - 1] as Stall;
                        for(int j = 0;
j<ked.makanan_tersedia; j++ )
                        {

        idkStall.listOfCookedFood1.Add(foodSatu);

        idkStall.listOfCookedFood2.Add(foodDua);

        idkStall.listOfCookedFood3.Add(foodTiga);
                        }

                        FMSView fmsView =
o.GetComponent<FMSView>();
                        fmsView.isFestival=true;
                        ObjScript objScript =
o.GetComponentInChildren<ObjScript>();
                        objScript.isFestival=true;
                        //GameObject p
=(GameObject)Instantiate(Loader.Load("Label -
status"));
                }
                jumlahPemain = i;
        }


        void ResetDataFestival ()
```

```
        {
                festivalSave = new InternalStorageUtil
("festivalSave.dat");
                festivalData = new FFMyFestivalData ();
                festivalData.idFestival = 0;
                festivalData.namaKedai = "";
                festivalData.idKedai = 0;
                festivalSave.Save (festivalData);
        }


        void UpdateStatusPemain ()
        {
                StartCoroutine ("LoadDataStatus");
                /*for (int i=0; i<listStatusPemain.Count;
i++) {
                        print (listStatusPemain[i]);
                }*/
        }

        IEnumerator LoadDataStatus()
        {
                //print ("MASOEKEKEK id
Kedai:"+kedaiPemain.idKedai);
                WWW www = new
WWW(service.LOADKEDAIFESTIVAL+festivalData.idFestival)
;
                float elapsedTime = 0.0f;
                while (!www.isDone) {
                        elapsedTime += Time.deltaTime;
                        if (elapsedTime >= 10.0f) break;
                        yield return null;
                }
                if (!www.isDone ||
!string.IsNullOrEmpty(www.error)) {
                        Debug.LogError(string.Format("Fail
Whale!\n{0}", www.error));
                        yield break;
                }
                string response = www.text;
```

```
                IList search = (IList)
Json.Deserialize(response);
                listStatusPemain = new ArrayList ();
                string namaF, uangF, statusF;
                foreach (IDictionary fest in search)
                {
                        namaF =
(string)fest["nama_kedai_festival"];
                        namaF = namaF.Replace("_", " ");
                        uangF =
(string)fest["makanan_tersedia"];
                        statusF =
(string)fest["status_pemain_festival"];
                        statusF = statusF.Replace("_", "
");
                        statusPemain = namaF+"\nMoney :
"+uangF+"\nMessage : "+statusF;
                        if(festivalData.namaKedai==namaF)
                        {
                                myStatus=statusF;
                        }

        listStatusPemain.Add(statusPemain);
                }
        }

        public Vector2 scrollPosition;

        void OnGUI()
        {
                //print ("list gambar :
"+listKedai.Count);

                //scrollPosition =
GUILayout.BeginScrollView(scrollPosition,
GUILayout.Width(sw), GUILayout.Height(sh));

                //GUILayout.EndScrollView();
        }
```

```
        void CreateStatusFont()
        {
                UILabel ux;
                for (int i=0; i<listStatusPemain.Count;
i++) {
                        p =
(GameObject)Instantiate(Loader.Load("Label -
StatusFont"));
                        ux =
p.GetComponentInChildren<UILabel>();
                        ux.text = listStatusPemain[i] as
String;
                        ux.transform.position = new
Vector3((float)LokasiStatusBoardX(i),
(float)LokasiStatusBoardY(i), 0);
                }
        }

        void UpdateTulisan ()
        {
                //print ("Up Jumlah : "+jumlahPemain);
                for (int i=1; i<=jumlahPemain; i++) {
                        //print
("Ganti:"+listStatusPemain[i-1] as String);
                        UILabel ux;
                        ux =
GameObject.Find("Status"+i).GetComponent<UILabel>();
                        //ux.text = "Haihai";
                        ux.text       =
listStatusPemain[i-1] as String;
                        //print("ux.text="+ux.text);
                }
        }


        int minutes=14,second=60,frame=0;
        string m, s;
        // Update is called once per frame
        void Update () {
                frame++;
```

```
            if (frame == 60) {
                    UpdateTulisan();
                    frame=0;
                    second--;
                    if(second==-1)
                    {
                            if(minutes==0)
                            {
                                    //permainan selesai
                                    ResetDataFestival();

        Application.LoadLevel("FoodFest");
                            }
                            minutes--;
                            second=59;
                    }
                    if(second%2==0)
                    {
                            UpdateStatusPemain();

        StartCoroutine("UpdateStatusFestival");
                    }
                    m = minutes.ToString();
                    s = second.ToString();
                    labelWaktu.text =
string.Concat(m,":",s);
            }
        }

        void OnClickSend (GameObject go)
        {
            if (msgStatus.text != "") {
                    myStatus = msgStatus.text;

        StartCoroutine("UpdateStatusFestival");
                    msgStatus.text = "";
            }
        }

        IEnumerator UpdateStatusFestival()
```

```
        {
                int uang =
Managers.Data.PlayerData.money;
                myStatus = myStatus.Replace (" ", "_");
                WWW www = new
WWW(service.UPDATESTATUSFESTIVAL+festivalData.namaKeda
i+"/"+festivalData.idFestival+"/"+uang+"/"+myStatus);
                float elapsedTime = 0.0f;
                while (!www.isDone) {
                        elapsedTime += Time.deltaTime;
                        if (elapsedTime >= 10.0f) break;
                        yield return null;
                }
                if (!www.isDone || !string.IsNullOrEmpty
(www.error)) {
                        Debug.LogError (string.Format
("Fail Whale!\n{0}", www.error));
                        yield break;
                }
                string response = www.text;
        }
}
```

Kode Sumber B.3 Kelas `FFGameplayManager`

DAFTAR PUSTAKA

[1] International Center for the History of Electronic *Game*s, "*Game* History," International Center for the History of Electronic *Game*s, [Online]. Available: http://www.icheg.org/icheg-*game*-history/timeline/. [Accessed 8 June 2014].

[2] N. Lovell, "*Game*s Brief," 01 2011. [Online]. Available: http://www.*game*sbrief.com/2011/01/what-is-a-social-*game*/. [Accessed 19 February 2014].

[3] Aingindra, "Android Adalah – Pengertian Android – Sistem Operasi," 25 December 2013. [Online]. Available: http://www.aingindra.com/android-adalah-pengertian-android-sistem-operasi.html. [Accessed 4 March 2014].

[4] Unity, "Unity3D," 2013. [Online]. Available: http://unity3d.com/pages/create-*game*s. [Accessed 18 February 2014].

[5] IBM, "Synchronous and asynchronous processes," IBM, 2000, 2004. [Online]. Available: http://pic.dhe.ibm.com/infocenter/adiehelp/v5r1m1/index.jsp?topic=%2Fcom.ibm.etools.ctc.bpel.doc%2Fconcepts%2Fcsynch.html. [Accessed 4 March 2014].

[6] S. Singh, "Synchronous Vs Asynchronous *gameplay* in online *game*s," Gamasutra, 14 February 2012. [Online]. Available: http://www.gamasutra.com/blogs/SiddharthSingh/20120214/91081/Synchronous_Vs_Asynchronous_*gameplay*_in_online_*game*s.php. [Accessed 4 March 2014].

[7] C. Janssen, "techopedia," [Online]. Available: http://www.techopedia.com/definition/27054/massively-multiplayer-online-*game*-mmog. [Accessed 25 June 2014].

[8] M. Weems, "Different Types of MMOs: MMORPGs, MMO First-Person Shooters, and MMO Real-Time Strategies," Altered *Game*r, 17 March 2017. [Online]. Available:

http://www.altered*game*r.com/pc-gaming/15232-different-types-of-mmos-mmorpgs/. [Accessed 26 June 2014].

[9] M. Rouse, "Web services (application services)," [Online]. Available: http://searchsoa.techtarget.com/definition/Web-services. [Accessed 26 June 2014].

BIODATA PENULIS

Muhammad Riduwan, lahir di Jombang pada tanggal 15 Januari 1992, merupakan anak pertama dari dua bersaudara. Penulis telah menempuh pendidikan mulai RA Miftakhul Ulum (1996-1998), SDN Kedungpari 3 (1998-2004), SMPN 1 Mojowarno (2004-2007), SMAN 3 Jombang (2007-2010), dan terakhir sebagai mahasiswa Teknik Informatika ITS (2010-2015). Selama masa perkuliahan di Teknik Informatika ITS, penulis aktif dalam kegiatan keorganisasian, diantaranya Himpunan Mahasiswa Teknik Computer-Informatika (HMTC), Studi Islam Teknik Computer (SITC), Keluarga Muslim Informatika (KMI) dan Kopma dr.Angka ITS. Penulis juga beberapa kali menjadi asisten dosen, diantaranya Asisten Dosen mata kuliah Statistika Komputasional dan Asisten Dosen mata kuliah Realitas Virtual. Penulis juga pernah menjadi finalis Gemastik kategori Sistem Cerdas pada tahun 2013. Dalam menyelesaikan pendidikan sarjana, penulis mengambil bidang studi Rekayasa Perangkat Lunak (RPL) dan pada tugas akhir penulis mengambil bidang studi Interaksi, Grafik, dan Seni (IGS). Penulis dapat dihubungi melalui email di mr.keuze@gmail.com.